QUADRIC-OF-REVOLUTION (QUADOR) BEAMS AND THEIR APPLICATIONS TO LATTICE STRUCTURES

A Thesis
Presented to
The Academic Faculty

By

Ashish Gupta

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Interactive Computing

Georgia Institute of Technology

August 2020

QUADRIC-OF-REVOLUTION (QUADOR) BEAMS AND THEIR APPLICATIONS TO LATTICE STRUCTURES

Approved by:

Prof. Jarek Rossignac, Advisor
School of Interactive Computing
Georgia Institute of Technology

Prof. Greg Turk
School of Interactive Computing
Georgia Institute of Technology

Prof. Athanassios Economou
School of Architecture
Georgia Institute of Technology

Prof. Concettina Guerra
School of Interactive Computing
Georgia Institute of Technology

Dr. Suraj Musuvathy
Lead Geometry Engineer
nTopology Inc.

Date Approved: July 2, 2020

ACKNOWLEDGEMENTS

"A PhD program is as much about building one's "character" as it is about building one's "personality"."

May not be these exact words, but something in this spirit is once conveyed to me by Prof. Jarek Rossignac, my PhD advisor. And not just in words, or spirit, to me he is the embodiment of such abstract qualities. He encourages his students to take up complex problems and with equal emphasis, rewards them for being elegant in their solutions. I will always remember his six 'C's'. Correct, Complete, Clever, Convincing, Clear, Concise, not sure if I got them all and in the right order. As words are not enough, and this thesis is just a milestone, I continue to thank him for being my teacher and learn from him.

I thank Prof. Greg Turk, and Prof. Karen Liu who enriched my work with their review and feedback throughout my PhD studies. I am equally grateful to them for broadening my knowledge in the field of Computer Graphics through their insightful teaching.

My gratitude to my committee members, Prof. Concettina Guerra, Prof. Athanassios Economou, and Dr. Suraj Musuvathy for their time and their invaluable feedback.

My big appreciation to the members of the Geometry Group and the Magic Group, specially to Kelsey Kurzeja, Mukul Sati, Sarang Joshi, Alex Clegg, and Wenhao Yu for stimulating discussions that I believe are vital for all-round development of a researcher.

I am indebted to my mother Rajdulari Gupta and my six elder sisters, who nurtured me like mother. I am forever grateful that I am raised by these seven women in a value system that makes good human beings.

Again, words are not enough to thank my wife Gowri, so, I will continue to thank her by contributing more in household chores. She juggles many hats, an accomplished engineer, a caring mother, a loving wife. Just as I learn from my advisor, I will continue to learn from her and improve.

Lastly but more importantly, my kids Siddharth and Sneha, I request them to not dis-

close it to my advisor that I improved some of my fundamentals with their help. I thank them with a big hug and yes, with some more playtime.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Additive Manufacturing (AM) or 3D printing presents the potential to fabricate parts with novel mechanical properties by allowing control over the internal microstructure of these parts. Lattices used for designing these microstructures are often defined by 3D arrangements of balls and beams. Attempting to discover optimal structures with desired mechanical properties, engineers explore variations: of the overall shape of these arrangements; of the dimensions and placements of the balls; of the profile shape of the beams and of the lattice connectivity that they form.

Current Solid Modeling technology only provides limited support for such explorations for several reasons: (1) Each beam connects two balls. It is often desired to ensure that the surface of the beam connects with tangential continuity to the surfaces of the two balls that it joins. Furthermore, it is often desired to use beams with curved profiles and not just those bounded by straight circular cone sections. The round-off errors resulting from computing the intersections of such beams, and hence the precise boundary representation of the lattice prohibit doing so reliably when the beams are relatively small compared to the overall size of the lattice. (2) Microstructure lattices have extremely high complexity (possibly billions of beams). Precise models of such structures are not supported by the current generation of commercial modellers. Furthermore, it is prohibitive to compute mass properties of such lattices by iterating over its balls and beams.

In this thesis, we propose to address both of these problems: (1) The challenge of designing and modeling beams with curved profiles and of computing their boundary representation reliably and accurately. (2) The challenge of precisely modeling highly complex lattices with profiled beams and of efficiently computing their mass properties.

To address the first challenge:

1. We propose a family of beams, which we call the ***quador beams***, that are bound by a quadric-of-revolution (quador) surface abutting tangentially to the two balls. This family

includes cone beams. We propose geometric constructions for quador beams, that have simple mathematical expressions and provides intuitive control of its shape.

2. We propose three analytically exact representations, Constructive Solid Geometry (CSG), Constructive Solid Trimming (CST), and Boundary Representation (Brep) of a lattice with quador beams. Each of these representations is more suitable than others in efficiently performing certain geometric queries on a lattice, e.g. CSG for Point Membership Classification (PMC), CST for ray intersection, and Brep for minimum distance query. We propose compact data structures to store these three lattice representations, describe efficient methods to compute them, and provide associated operators to efficiently query the lattice.
3. We propose a numerically robust approach to compute the topology of the Brep a lattice of quador beams. In this approach we infer the topology of the Brep of the lattice from the Brep of a polyhedral complex. This complex is formed by a set of linear half-spaces, each of which trims a quador (or a ball) surface of the lattice. Our approach avoids computing square roots (or any other irrational values) and involves only rational numbers and basic arithmetic operations (addition, subtraction, multiplication and division) on them.

To address the second challenge:

1. We propose a class of lattices, which we call the ***Steady Lattices***. A steady lattice consists of a three-directional tensor of cells of balls and beams that each connects two balls in the same cell or in two different cells. We discuss rows of balls and beams in each direction. In a steady lattice, all rows in at least one direction are steady, by which we mean that each cell of balls in a row and the beams incident on its balls are related to the previous cell by the same similarity transformation. We use an extremely concise representation of the lattice: The balls and beams of a template cell, three transformations, and 3 repetition counts. Steady lattices are easy to compute and are amenable to fast and robust queries.

2. We propose closed-form expressions that exploit steadiness for computing mass properties (e.g., surface area, volume, center-of-mass) of a steady lattice in constant time per steady row of cells of balls and their incident beams, i.e, without iterating over all of the elements of the lattice.

Our work lays the foundation for designing enormously large (trillions of beams) and complex (bent, graded and complex connectivity of beams) lattice microstructures for 3D printed parts and forms the backbone of the state-of-the-art geometric applications developed for DARPA's TRAansformative DESign (TRADES) project. Part of the research reported in this thesis is published in the following papers [1, 2, 3, 4]:

[1] A. Gupta, G. Allen, and J. Rossignac, Quador: Quadric-of-revolution beams for lattices, Computer-Aided Design, vol. 102, pp. 160–170, 2018.

[2] A. Gupta, G. Allen, and J. Rossignac, Exact representations and geometric queries for lattice structures with quador beams, Computer-Aided Design, vol. 115, pp. 64–77, 2019.

[3] A. Gupta, K. Kurzeja, J. Rossignac, G. Allen, P. S. Kumar, and S. Musuvathy, Programmed-lattice editor and accelerated processing of parametric program-representations of steady lattices, Computer-Aided Design, 2019.

[4] W. Yaohong, A. Gupta, K. Kurzeja, J. Rossignac, CHoCC: Convex Hull of Cospherical Circles and Applications to Lattices, Computer-Aided Design, 2020.

CHAPTER 1
INTRODUCTION

Objects fabricated by Additive Manufacturing (AM) owe their unprecedented mechanical properties [5] to their architected internal microstructure [6]. Lattices offer a popular design space for exploring and optimizing this internal microstructures [7, 8, 9]. Indeed, lattices support a wide range of material structures and yet, because of the simplicity of their elements, provide for an efficient analysis of the overall aggregate properties and behavior [10].

In this thesis, we discuss algorithmic solutions for designing, modelling, displaying, and querying lattices. Specifically, the tools proposed here improve the ability of a scientist or an engineer to process, analyze, optimize, and print lattices of unprecedented complexity (Fig. 1.1) and hence to explore and produce novel material structures with unprecedented physical characteristics.

We define a lattice as a connected solid that can be expressed as the union of balls and beams. The balls are pairwise disjoint and each ball is bounded by a sphere. Each beam connects (i.e., is incident on) two of these balls. By controlling the position and size of balls, and the thickness of the beams, the lattice may be optimized for an intended application. Although we briefly discuss more general options, we focus on beams that are each a ***solid-of-revolution*** (a solid obtained by rotating a plane profile curve around a straight axis) around the axis containing the centers of its balls and that is bounded by three or four faces, two of which are planar and the other lie on quadric surfaces. Although in some applications, all balls may have the same radius, we focus on the more general cases where they do not.

A simple version [11, 12] of the lattices defined above is where the beams are cone beams (Fig. 1.2-left). As evident, in such a lattice, several beams are incident on a single

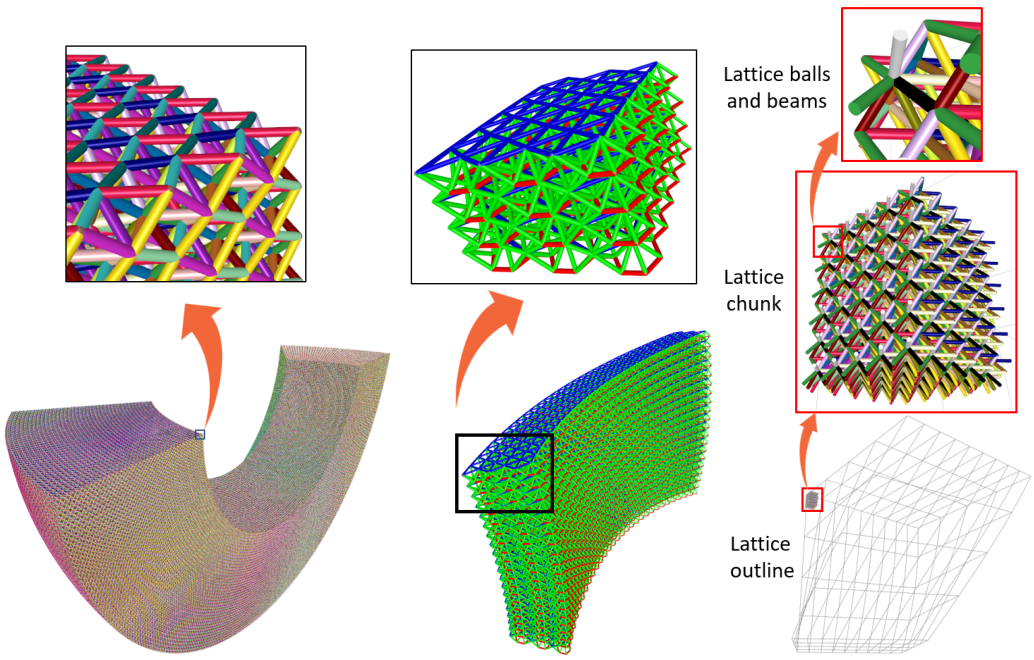


Figure 1.1: Large and complex lattices: (left) Octet lattice - 7,680,000 beams, (middle) Honeycomb-dual lattice - 64,800 beams, (right) octahedral lattice - more than 24×10^{12} beams.

ball. Hence even in a carefully designed lattice where two beams not incident on the same ball are disjoint, the beams incident on the same ball may intersect. Such intersections generate complex intersection curves (degree 4 curves in this case) and are difficult to represent precisely and to process for queries on the lattice. Furthermore, these complex intersections make it even more difficult to efficiently and closely estimate the surface area and volume of such lattices, which is already difficult owing to the large number of balls and beams in the lattice. One possible solution to circumvent some of these issues is to inflate each ball so as to contain the intersection of its incident beams (Fig. 1.2-right). Then all intersection curves are circles and it is possible to compute the exact surface area and volume of the lattice, as beams are disjoint right circular cone frusta and balls are spheres minus disjoint spherical caps. But adding inflated balls adds unnecessary weight to the lattice, more so when the angle between the beams incident on a ball is small. Furthermore, the isolated beam-ball junctions may form structurally weak spots. To summarize, even for simple lattices of sphere balls and cone beams, it is challenging to compute the precise

boundary (faces, edges, vertices) of the lattice, and to precisely compute the mass properties (surface area, volume) of the lattice.

In this thesis, we propose to address both of these problems: (1) The challenge of designing and modeling beams with curved profiles and of computing their surface reliably and accurately. (2) The challenge of precisely modeling highly complex lattices with profiled beams and of efficiently computing their mass properties.

We focus on osculating (kissing) beams with special properties (discussed in Ch. 4), where the quadric surface(s) of a beam connects tangentially (with normal continuity) to the surfaces of the two balls connected by that beam.

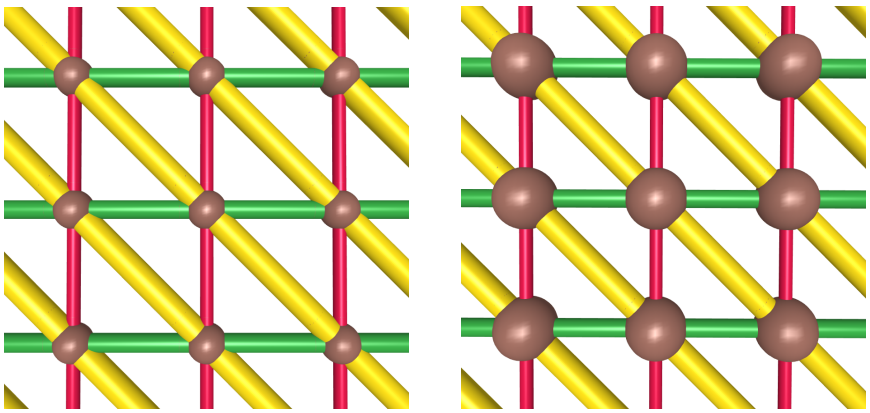


Figure 1.2: The subfigures above show simple lattices with cone beams. In the left subfigure, the intersections between surfaces of beams incident on a common node are quartic (degree 4) intersection curves. In the right subfigure the balls are inflated so as to contain the intersection between beams. Now the intersection curves are all circular but the weight of the lattice has increased considerably and the junction where a beam meets the inflated ball may become a structurally weak spot.

In Chapter 2, we discuss simplest version of such osculating beams called cone beams (right circular cone-frusta) (Fig. 1.3). We review simple closed-form expressions for the computing a convenient representation of a cone beam from the centers and radii of its two balls.

Although osculating cone beams are sufficient for a broad set of applications, lattice design and optimization may sometimes be improved by providing support for more general beams.

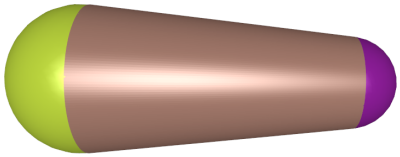


Figure 1.3: A cone beam (brown) between two spheres (green and magenta). The surface of the cone (brown) connects smoothly to the surfaces of the two balls.

In Chapter 3, we discuss beams that are bound by a quadric-of-revolution surface defined by a conic profile. We call these beams ***quador beams***, where "quador" stands for "quadric-of revolution". The cone beams discussed in the previous chapter are the simplest version of quador beams, wherein the profile curve is a straight-line. We discuss other options that we considered and our rationale for advocating quador-beams over them. We discuss an intuitive scheme to define and control the shape of a quador beam and provide associated analytical expressions that are remarkably simple. We also discuss ***biquador beams***, that are similarly simple to describe, yet are more versatile than quador beams. While a quador beam has a single quador surface, a biquador beam has two tangentially connected quador surfaces that bound the beam. Finally we discuss benefits of quador and biquador beams over simple cone beams.

Figure 1.4: The left subfigure shows a quador beam (brown) connecting two spheres (green and magenta). The quadric surface of the beam (brown) connects smoothly to the surfaces of the two balls. The right subfigure shows a biquador beam connecting two spheres. The biquador beam is bound by a left quador surface (light brown) which connects smoothly to the left ball, and a right quador surface (light magenta) that connects smoothly to the right ball, and the two quador surfaces connect smoothly to each other.

After describing quador and biquador beams, we bring the focus back to the lattices of such beams, wherein the challenge is to compute the intersections of the surfaces of these beams, reliably and accurately.

In Chapter 4, we first introduce a new entity that we call ***a hub*** and, discuss decom-

posing a lattice of quador beams into an assembly of hubs. A hub is defined as the union of a ball and its incident half-beams (one half of a beam that is split into two halves) (Fig. 1.5-right). We discuss special properties of a hub, specifically that the intersection of the surfaces of two half-beams of a hub is planar and is in fact a conic curve. Hence if we assume that the lattice is ***clean***, i.e., the hubs of the lattice are disjoint (other than two hubs touching at disks where a common beam between the two hubs is split), all intersection curves in a lattice of quador beams are conic curves. Conic curves have both implicit and parametric representations and are much simpler to work with than degree 4 intersection curves of two quadric surfaces in general.

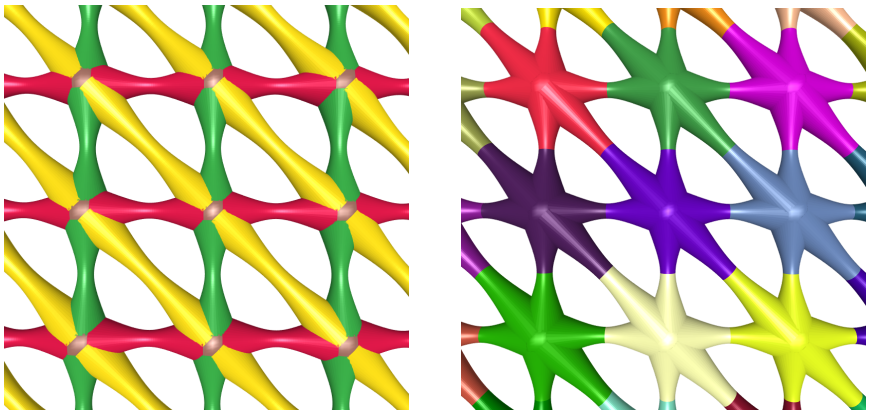


Figure 1.5: The subfigure on the left shows a portion of a lattice with quador beams. Here we consider a lattice as the union of balls and beams. We split each beam into two half-beams and define an entity called "hub", as the union of a ball and its incident half-beams. The subfigure on the right shows the same portion of the lattice as in the left subfigure, but now we show the lattice as an assembly of hubs, each in a different color.

Having a lattice with curved profile beams (quador beams) and all intersection curves as conics is remarkable. What we need now is a lattice representation, that is compact, have exact curves and surfaces instead of their polygonal approximations, and supports efficient processing of queries on the lattice.

In Chapter 5, we discuss computing three different representations of a hub. Extending these representations to a clean lattice, i.e. an assembly of disjoint hubs is trivial. We discuss 1) the ***CSG (Constructive Solid Geometry)*** representation wherein a hub is the union of its balls and half-beams, 2) the ***CST (Constructive Solid Trimming)*** representa-

tion, which is the union of solid chunks, each chunk corresponds to a beam (or the ball) of the hub and is a quador half-space trimmed by several linear half-spaces (trimming polyhedron), and 3) the ***Brep (Boundary representation)***, i.e. the geometry and the connectivity of the faces, edges and vertices of the hub (Fig. 1.6). We discuss ***corner table data structure*** [13] to compactly store and efficiently operate on these representations.

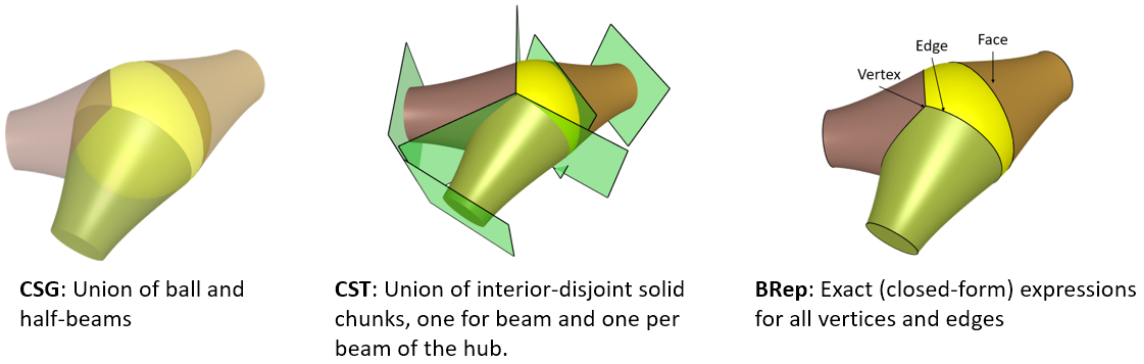


Figure 1.6: Three representations of a hub. Each one is more suitable to perform certain queries on the hub than the others.

Although exact in terms of the nature of the geometry (analytical as opposed to polygonal approximation) of the underlying curves and surfaces, the representations discussed above may suffer from topological inaccuracies due to numerical round-off errors, e.g. a candidate intersection of three surfaces of a hub may be wrongly classified as a vertex of the Brep of that hub when it is not, as it lies inside other beams of that hub.

In Chapter 6, we discuss a simple and numerically robust method to compute the correct topology of a hub. Our method exploits the fact that we can associate a polyhedral complex (Fig. 1.7) to a hub, each room of this complex corresponds to the trimming polyhedron of a beam (or the ball) of the hub. We first discuss numerically robust computation of a ***dart*** [14] based representation of the trimming complex of the hub and then write the operators to traverse the boundary (faces, edges, vertices) of that hub indirectly in terms of the dart operators of its trimming complex. In our simplified approach, we only need to deal with signs of numbers with non-nested square roots and we discuss efficient ways to compute that precisely.

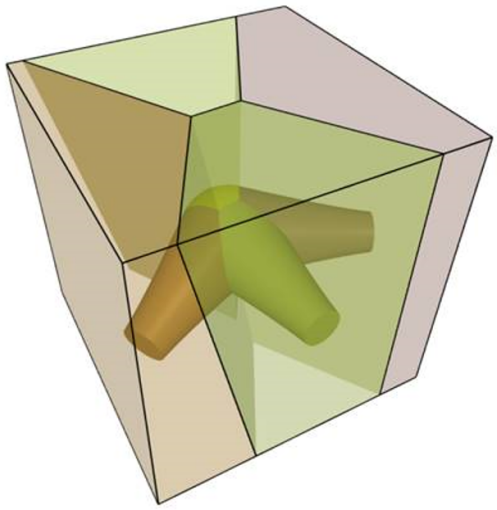


Figure 1.7: A hub and its polyhedral trimming complex. Each room of the complex (a polyhedron) trims a beam (or the ball) of the hub.

Once we have numerically robust representation of hubs with profiled beams, the next challenge that needs to be addressed to support extremely large and complex lattices (Fig. 1.1) is the fast and accurate processing its hubs. This requires representing the lattice and therefore its hubs in a form that is easy to compute, compact to store and is amenable to fast and robust queries.

In Chapter 7 we propose a class of lattices that we call ***steady lattices***. A steady lattice is defined by a small program (approx. 50 lines), can smoothly bend, grade (thickness of beams) and twist, has only quadric surfaces and conic curves, and only a few control parameters. All these properties makes steady lattices extremely efficient for designing large and complex lattice microstructures. The lattices shown in Fig. 1.1 are all steady lattices. The research on steady lattices is done collaboratively with my colleague Kelsey Kurzeja. In this thesis, I state the definition of steady lattices (which stems from this collaborative effort), as that is necessary for clarity, besides that I report only my individual contributions on steady lattices.

A common objective of modeling the interior of a 3D printed part with lattice microstructures is to increase the strength to weight ratio of the part, i.e. to reduce the overall

weight of the part while maintaining required strength. For other applications (e.g. heat transfer, adsorption), it may be required to increase the surface area to volume ratio of the lattice. To achieve these objectives, it may be required to compute the mass properties, such as the surface area or the volume of the lattice microstructure. It may not be viable to compute the surface area or volume of such a large lattice by iterating over its billions of elements.

In Chapter 8, we provide closed-form expressions for accelerated computation of the mass properties, i.e. the surface area, the volume, and the center of mass of a steady lattice. We exploit the special property of a steady lattice that it consists of ***steady rows of hubs***, by which we mean that a hub in that row is related to the previous hub in that row by the same similarity transform. Thus the mass property, e.g. the volume of all the hubs in that row is computed from the volume of the first hub in that row, the similarity transform of that row, and the number of hubs in that row. We provide closed-form expressions to do so and discuss a simple approach to efficiently and closely approximate the mass properties of a single hub.

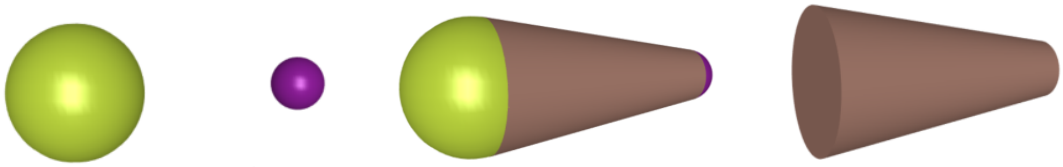
Finally in Chapter 9, we discuss advantages of special properties of quador beams and of representations of lattices of quador beams to efficiently perform fundamental geometric queries, such as PMC, minimum distance, ray intersection, slicing, volume meshing on a lattice. For example, for a clean lattice which is an assembly of non-interfering hubs, a query may be distributed (for parallel processing) over all the hubs of that lattice. Furthermore, for a clean steady lattice, we may first efficiently identify a small set of candidate hubs [15] and then distribute the query only on those hubs. Thereafter, for a single hub, we choose the most suitable representation (CSG,CST, or Brep) of that hub to perform that query.

We conclude in Chapter 10, summarizing our contributions and discussing future directions for research.

CHAPTER 2

CONE BEAMS

As mentioned in the introduction, our focus in this thesis is on beams that each connects two balls, is a solid-of-revolution with axis passing through the centers of the two balls, and is bounded by a quadric-of-revolution surface that connects tangentially to the surfaces of the two balls. The simplest of these beams is what we call a ***cone beam*** [16]. A cone beam is a right circular cone frustum, with axis of the cone passing through the centers of the two balls and the surface of the cone connects tangentially to the surface of the two balls (Fig. 2.1).



(a) Two balls

(b) Two balls with a cone beam between them

(c) The cone beam

Figure 2.1: A cone beam (brown) between two spheres (green and magenta). It is a right circular cone frustum. The surface of the cone (brown) connects smoothly to the surfaces of the two balls.

A cone beam is defined implicitly from the centers and radii of its two balls. In the next section, we provide simple closed-form expressions to construct a cone beam.

2.1 Construction of a cone beam

Given two balls with centers (C_1, C_2) and radii (r_1, r_2) (Fig. 2.2), we consider the cross-section in a plane passing through the centers of the two balls. In that cross-section we obtain a circle from each of the two balls. We compute an outer tangent PQ (tangent that does not intersect the line joining the centers of the two circles) using the following

expressions. Please refer to Appendix A for notations used in the equations below:

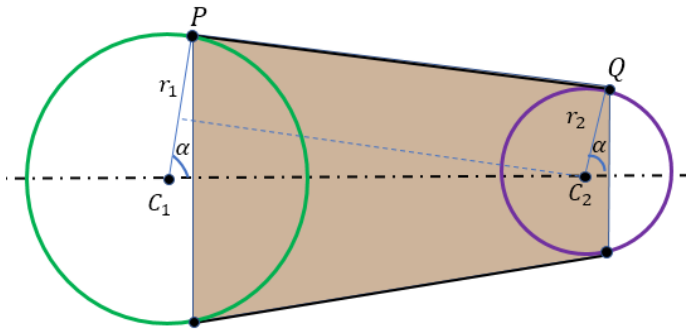


Figure 2.2: Given two balls, the figure above shows the cross-section in a plane passing though the centers of the two balls. We compute the common tangent PQ and revolve it around the axis (dotted line) passing through the two ball centers to construct the surface of the cone beam. The shaded (brown) region corresponds to the cone beam.

$$
\begin{aligned}
P &= C_1 + r_1 \cos(\alpha)\overrightarrow{C_1C_2} - r_1 \sin(\alpha)\overrightarrow{C_1C_2}^{\circ} \\
Q &= C_2 + r_2 \cos(\alpha)\overrightarrow{C_1C_2} - r_2 \sin(\alpha)\overrightarrow{C_1C_2}^{\circ} \\
&\text{where, } \cos(\alpha) = \frac{r_1 - r_2}{|\overrightarrow{C_1C_2}|}
\end{aligned}
\tag{2.1}
$$

We then revolve the profile, i.e. the line segment PQ around the axis passing through the centers C_1 and C_2 to create the surface-of-revolution of the cone beam. This surface touches the surfaces of each of the two balls in a contact circle. The region bounded by this cone surface and the disks corresponding to the two contact circles is the solid-of-revolution that we call a cone beam. Note that the union of the two balls and the cone beam is the convex hull of the two balls.

As described above, cone beams are simple to construct. Furthermore, they being conic frustum, a solid primitive, are well supported by CAD packages and are becoming one of the common beam formats for modeling lattice structures in 3D printing industry [17]. Fig. 2.3 shows a lattice with cone beams.

Though sufficient for many applications, cone beams are limiting in design freedom. As a cone beam is implicitly defined by its two balls, there is no degree of freedom left

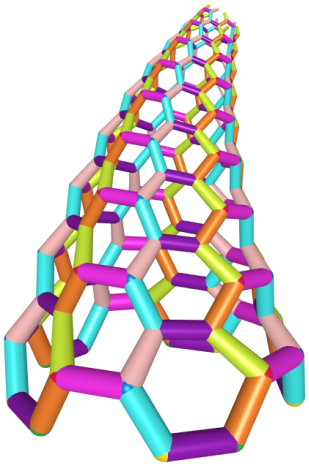


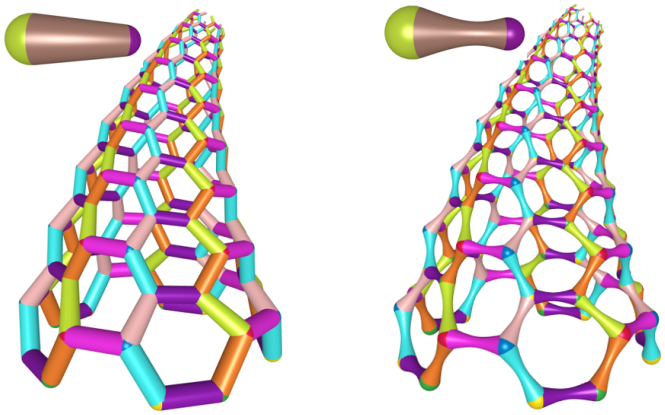
Figure 2.3: A lattice with cone beams

to control the shape of the beam independent of the two balls. This severely restricts an engineer's ability to optimize the lattice. For example, growing a ball will thicken all beams incident on that ball, which may not allow engineers to create thinner and thicker beams along specific directions. In the next chapter, we propose a new family of beams that like cone beams are simple to construct and process, but offer more design freedom than cone beams.

CHAPTER 3

QUADOR BEAMS

In the previous chapter we discussed the simplest version of solid-of-revolution (a solid obtained by rotating a plane profile curve around a straight axis) beams between two balls, that we called the cone beams. For a cone beam the profile curve is a straight line. In this chapter we propose a family of solid-of-revolution beams, wherein the profile curve is a conic and is symmetric about the axis-of-revolution. The surface thus generated is a quadric-of-revolution surface [18], which in short we call a ***quador*** surface, and the corresponding beam a ***quador beam*** [1]. Fig. 3.1b shows an example of a quador beam and a lattice constructed from quador beams.



(a) (b)

Figure 3.1: A cone beam and a lattice with cone beams (left). A quador beam and a lattice with quador beams (right).

When selecting the mathematical nature of the curved beams that should be supported in a CAD/CAM system, one may consider the following metrics: (1) The range of possible beam-shapes that it supports, (2) The possibility of exposing a small set of intuitive parameters for controlling the beam-shape precisely, (3) The mathematical complexity, computational cost, and numerical accuracy of queries, such as Point-Membership Classification,

Ray-Casting, or Boundary Evaluation.

In Section 3.1, we define our metrics more precisely, and use them to assess prior art and other alternatives that we have considered. In Section 3.2, we propose intuitive and effective control parameters for ***quador beams*** and provide closed-form expressions for computing their precise geometry. We also illustrate the range of shapes that quador beams offer. In Section 3.3, we propose an extended family of quador beams which have not one but two quador surfaces that each connects smoothly to a ball and the two surfaces connect smoothly to each other. We call these beams ***biquador beams***. We propose an intuitive parameterization of ***biquador beams*** and provide details of their construction.

3.1 Prior art and alternatives explored

Given circles $\mathbf{C_1}$ and $\mathbf{C_2}$ (Sec. 2.1), we present several options for a profile $\mathbf{C}$ that connects to $\mathbf{C_1}$ and $\mathbf{C_2}$ with tangential continuity (Fig. 3.2). $\mathbf{C}$ is invalid if it crosses axis $\mathbf{A}$ passing through the centers of $\mathbf{C_1}$ and $\mathbf{C_2}$.

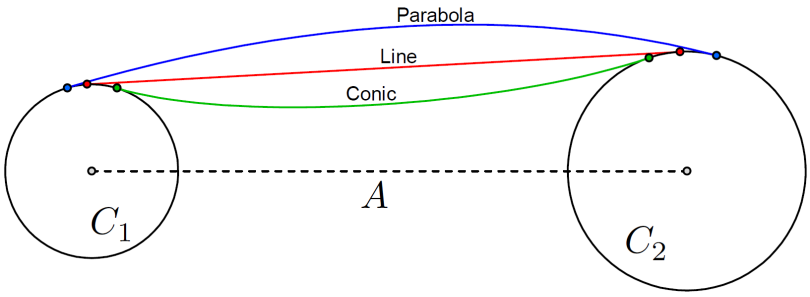


Figure 3.2: Simple Profile Curves: Line (red), Parabola (blue), general Conic (green).

Line: We have discussed this case (Fig. 3.3) in Chapter 2. Max [19] called them "cone-spheres" and used them to approximate and render tubular structures. Note that this solution has no degree-of-freedom and hence does not offer any control parameter.

Conic: Given two points and associated tangent directions, we can construct a conic section curve $\mathbf{C}$ that smoothly connects the circles $\mathbf{C_1}$ and $\mathbf{C_2}$ (green curve in Fig. 3.2). Equivalently, we can think of $\mathbf{C}$ as a rational quadratic Bézier curve. There is one free

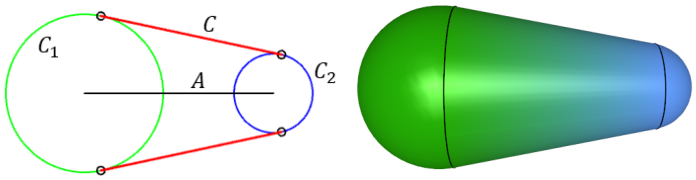


Figure 3.3: Cross-section (left) with profile curve (red), axis (black) and beam (right).

parameter, which we can fix by specifying a conic shape parameter or (equivalently) the weight assigned to the middle control point in the rational Bézier form. One option is to set this weight equal to 1, which produces a parabola (blue curve in Fig. 3.2). Revolving such a conic around the axis of the beam yields a surface defined by an implicit equation of degree 4.

Symmetric conic: We can restrict $\mathbf{C}$ to be a conic of which axis-of-symmetry is the axis-of-revolution of the beam. The resulting surface is a quadric-of-revolution (i.e. a quador) surface, so it has an implicit equation of degree 2. Quadrics-of-revolution provide a rich set of primitives for Geometric and Solid Modeling and have been studied by Goldman [18] and others. Most relevant to our work is the paper by Jia et al. [20], which calls them "Revolute Quadrics" (abbreviated RQ) and proposes closed-form expressions for computing their profile curves. Fig. 3.4 shows hyperbolic and elliptic profiles and the resulting quador beams.

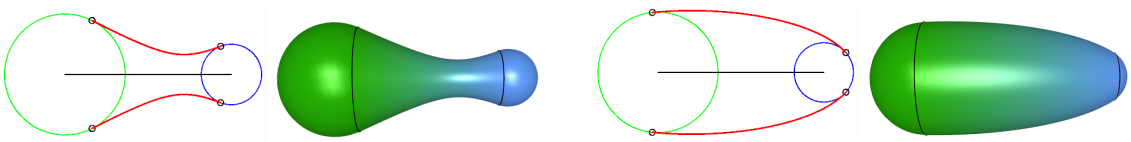


Figure 3.4: Quador beams.

Circular arc: In this construction, curve $\mathbf{C}$ is a circular arc tangent to the circles $\mathbf{C_1}$ and $\mathbf{C_2}$, as shown in Fig. 3.5, and the resulting surface $\mathbf{S}$ is a portion of a torus. If the beam shape is convex, the toroidal surface is self-intersecting and we use a subset that is inside, and not on the boundary of, the solid torus.

The circular arc $\mathbf{C}$ has one remaining degree of freedom, which can be fixed in a variety

of different ways. For example, we may specify the radius of $\mathbf{C}$, or a "bulge" or "sag" parameter, as in [21]. Alternatively, we may require $\mathbf{C}$ to pass through a given point, or to be tangent to a third circle.

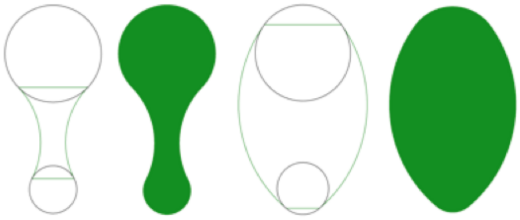


Figure 3.5: Circular Arc beams.

Biarc: If we choose contact points on the circles $\mathbf{C_1}$ and $\mathbf{C_2}$, then we have two points and two tangent directions that the curve $\mathbf{C}$ must interpolate. There are several different types of curves that can be used, as discussed in the next few sections. One interesting option is a biarc [22, 23], which consists of two circular arcs that join tangentially. The biarc curve has one remaining degree of freedom that can be specified in several different ways, as described in [24]. Three examples are shown in Fig. 3.6. The resulting beam is composed of two smoothly joined torus pieces.

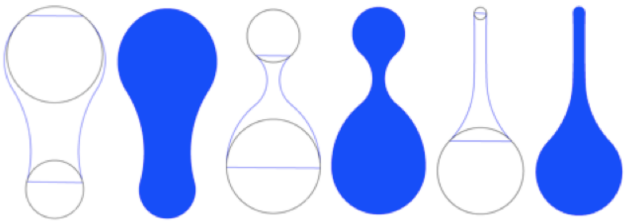


Figure 3.6: Biarc beams.

Symmetric biconic: Another option is to use two symmetric conic curves that join each other with tangent continuity. The resulting beam consists of two smoothly connected quador surfaces, so we call it a ***biquador*** beam. Some examples are shown in Fig. 3.7.

Parametric cubic: Instead of a biarc, we may use a parametric cubic curve to interpolate the point and tangent end-conditions. Glassner [25] proposed to specify $\mathbf{C}$ as a cubic Hermite curve controlled using the point at which the tangent lines cross or using

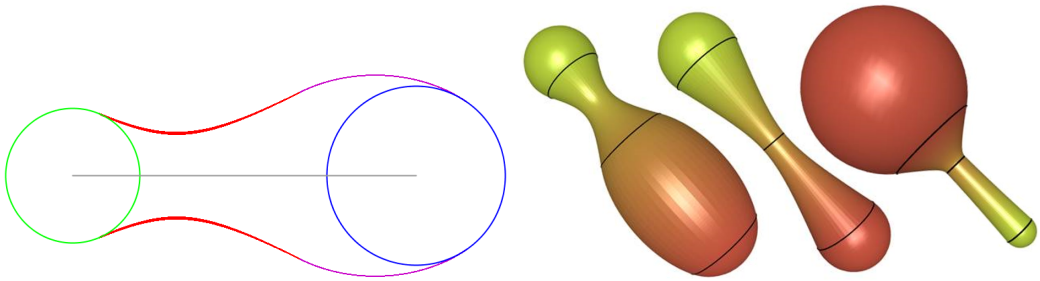


Figure 3.7: Biquador beams.

two points to define a Bézier control polygon.

Parametric quintic curve: We explored defining $\mathbf{C}$ as a quintic parametric curve that connects to both circles with second degree continuity. Fig. 3.8 exhibits the diversity of this family.

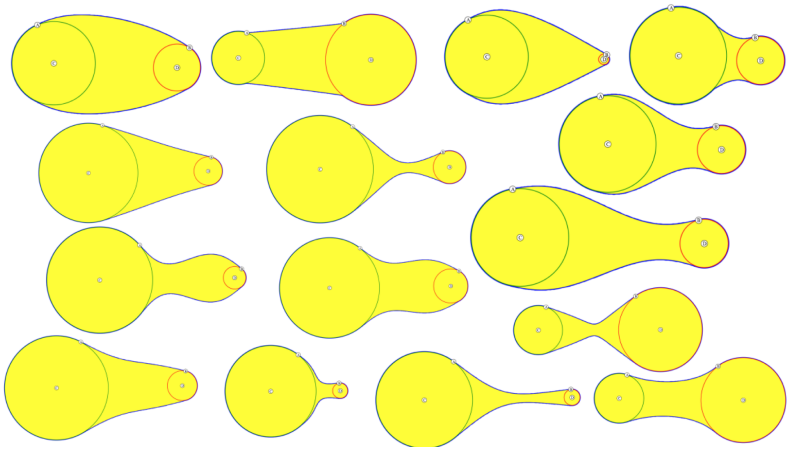


Figure 3.8: Quintic beams.

3.1.1 Rationale for advocating quador and biquador beams

In this section, we compare the design flexibility and computational cost of various beam shapes, and explain why we advocate the use of quador and biqador beams.

Ease of Computation: In general, if the profile curve can be represented by an implicit equation of degree n, then the resulting surface-of-revolution will be an implicit surface of degree $2n$. However, if the profile curve is symmetric about the axis-of-revolution, then the degree of the surface is reduced to just n [26]. So, we achieve a significant reduction in

surface degree by using symmetric conics to define quador and biquador beams.

The degree of the surfaces has a direct impact on the cost of doing basic computations on a beam. For example, to perform ray-casting on a surface of degree n, we must calculate the roots of a polynomial of degree n. If $n = 2$, this is trivially easy. If $n = 4$, the roots can be calculated from closed-form formulae. If $n \geq 5$, numerical methods are required. Similar arguments apply when computing planar cross-sections for traditional additive manufacturing processes: cross-sectioning quador surfaces is easy, as outlined in Section 4.2.2, but cross-sectioning surfaces of higher degree is considerably more difficult.

Flexibility: To compare various beam formulations on the flexibility of designing a variety of beams, we examine the number of degrees of freedom that remain in the curve $\mathbf{C}$ after we have constrained it to be tangent to the circles $\mathbf{C_1}$ and $\mathbf{C_2}$. Fig. 3.9 summarizes the considerations of flexibility versus ease of computation that we have discussed above.

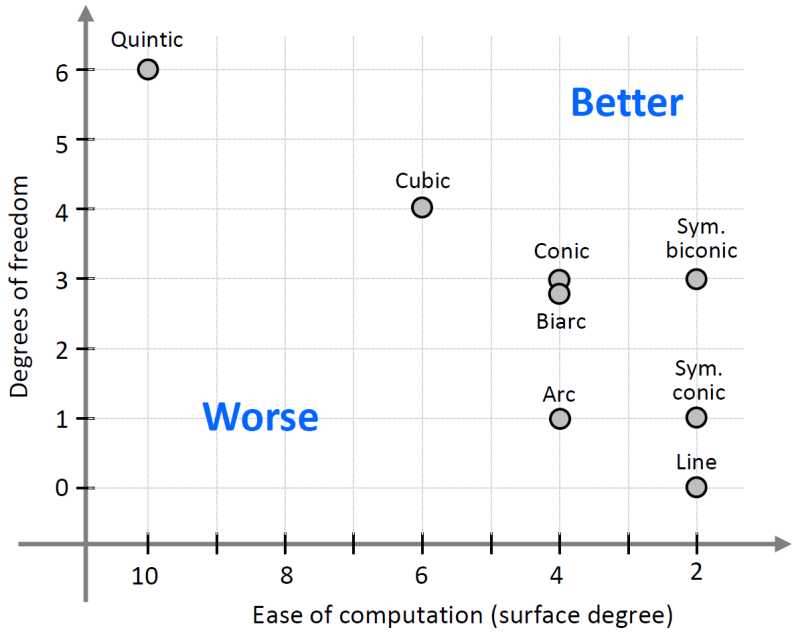


Figure 3.9: For each type of profile curve, we indicate its design flexibility (degrees of freedom) and the degree of the resulting surface-of-revolution.

As the chart shows, quador and biquador beams provide adequate design flexibility with computational costs that are no higher than simple cylinders and cones.

Boundary evaluation: When computing a Brep for some portion of a lattice, we have

to intersect pairs of quador surfaces that are tangent to a given ball. As explained in Section 4.2, the intersection of two quador surfaces that are tangent to a common sphere are conic (planar) curves. Furthermore, each vertex of the Brep is the intersection of a line with a quadric surface. So, the Brep may be obtained using simple closed-form calculations. Other beam representations involve more complex intersection curves. These can either be represented procedurally, which incurs large computational costs, or by spline approximations, which require careful handling of approximation tolerances.

Overall, quador and biquador beams provide a good trade-off between design flexibility and ease of computation, and, when joined at common tangent spheres, they allow for simple and efficient boundary evaluation. So, in a geometric modeling system based on these types of shapes, we would expect the computational code to be small, fast, reliable, and relatively easy to develop. Small code size has a major impact on development costs. In addition, small size is an important factor in cloud architectures, where code is often transferred across wide-area networks.

3.2 Construction of a quador beam

Jia et al. [20] computed the implicit equation of the symmetric conic in a frame with origin at the center of one of the circles and the axis-of-revolution as its x-axis. Our approach is similar, but we use a somewhat different frame to exploit symmetry and keep the number of parameters to a minimum.

3.2.1 Construction in a symmetric frame

We use a coordinate system as shown in Fig. 3.10, which places the centers of the circles $\mathbf{C_1}$ and $\mathbf{C_2}$ at coordinates $(-1, 0)$ and $(1, 0)$ respectively. Without loss of generality, we may assume that the radii of the two circles satisfy $r_1 \leq r_2$.

Any conic section curve (ellipse, hyperbola, or parabola) that is symmetric about the

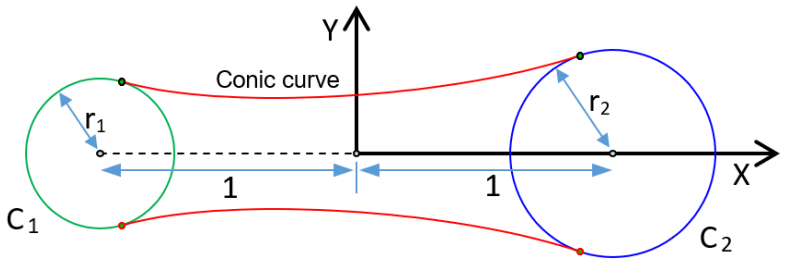


Figure 3.10: Construction in a symmetric frame.

x-axis can then be represented by a quadratic equation in the general form:

$$ax^2 + y^2 + 2bx + c = 0 \tag{3.1}$$

We need to find values of a, b, c that will make this curve tangent to the two given circles $\mathbf{C_1}$ and $\mathbf{C_2}$. Tangency to the two circles will fix only two of these three parameters. So, for the time being, we let a be a free input variable that controls the shape of the conic, and we calculate b and c as functions of a. Later, we will replace a by a different free parameter that has a more clear geometric meaning.

To further exploit symmetry, we define $r = (r_2 + r_1)/2$ and $d = (r_2 - r_1)/2$, i.e. $r_1 = r - d$ and $r_2 = r + d$. With these definitions, the equations of our two circles become:

$$\mathbf{C_1}: \quad (x+1)^2 + y^2 = (r-d)^2 \tag{3.2}$$

$$\mathbf{C_2}: \quad (x-1)^2 + y^2 = (r+d)^2 \tag{3.3}$$

To find the points where the general conic intersects circle $\mathbf{C_1}$, we eliminate y^2 from (3.1) and (3.2). This gives us

$$ax^2 + 2bx + c = (x+1)^2 - (r-d)^2 \tag{3.4}$$

If the conic and circle are tangent, then this equation will have repeated roots, so it's discriminant will be zero. So, we equate the discriminant to zero, and solve for c. Then,

after substituting $m = a - 1$, for brevity, we get:

$$c = \frac{1}{m}\left[(b-1)^2 - m(r-d)^2 + m\right] \tag{3.5}$$

Similar reasoning with circle $\mathbf{C_2}$ gives:

$$c = \frac{1}{m}\left[(b+1)^2 - m(r+d)^2 + m\right] \tag{3.6}$$

We equate the two expressions for c from (3.5) and (3.6), and solve for b:

$$b = dmr \tag{3.7}$$

We can substitute this value of b back into either (3.5) or (3.6) to get c as a function of m alone:

$$c = \left(1 + \frac{1}{m}\right) + \left(mr^2 - 1\right) d^2 - r^2 \tag{3.8}$$

If we substitute b from (3.7) and c from (3.8) back into (3.4), we get an equation that we can solve to obtain the x-coordinate of the tangency points on circle $\mathbf{C_1}$. Straightforward calculations show that these tangency points (x_1, y_1) are given by

$$x_1 = -\frac{1}{m}(dmr - 1) \quad ; \quad y_1 = \pm\sqrt{r_1^2 - (x_1 + 1)^2} \tag{3.9}$$

Similar computations show that the tangency points (x_2, y_2) on circle $\mathbf{C_2}$ are given by

$$x_2 = -\frac{1}{m}(dmr + 1) \quad ; \quad y_2 = \pm\sqrt{r_2^2 - (x_2 - 1)^2} \tag{3.10}$$

3.2.2 Control of a beam's shape by parameter a of its symmetric-conic profile

Jia et al. [20] discusses the correspondence between the sign of parameter a and of another parameter $g = b^2/a - c$. However, they do not specifically state the minimum and maximum

values of a that bound the valid set of profiles. By valid we mean, a profile that is tangent to the two given circles and produces a singly connected solid-of-revolution. For example, Fig. 3.11 shows two invalid profiles.

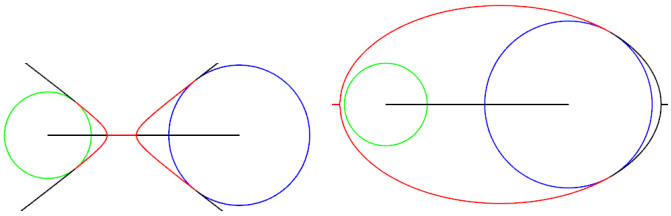


Figure 3.11: Invalid profiles for constructing a quador beam. The profile in the left subfigure does not produces a singly connected solid-of-revolution and the profile in the right subfigure looses contact with the green circle.

The table below and Fig. 3.12 shows the types of beam shapes corresponding to various values of a

Table 3.1: Values of a and corresponding beam type

Value of a	**Color**	**Beam Type**
$a_{min} = \dfrac{r^2}{r^2-1}$	Green	Cone (hourglass)
$\dfrac{r^2}{r^2-1} < a < \dfrac{d^2}{d^2-1}$	Cyan	Hyperbola
$a = \dfrac{d^2}{d^2-1}$	Black	Cone
$\dfrac{d^2}{d^2-1} < a < 0$	Pink	Hyperbola
$a = 0$	Magenta	Parabola
$0 < a \leq \dfrac{dr+d-r}{dr+d-r-1}$	Orange	Ellipse
$a_{max} = \dfrac{dr+d-r}{dr+d-r-1}$	Red	Limit ellipse

Note that the pink colored hyperbola in Fig. 3.12 is a horizontally opening hyperbola with its single branch forming the beam (also shown explicitly in Fig. 3.13). The limiting ellipse described in the last row of the table touches the smaller of the two circles at a single point.

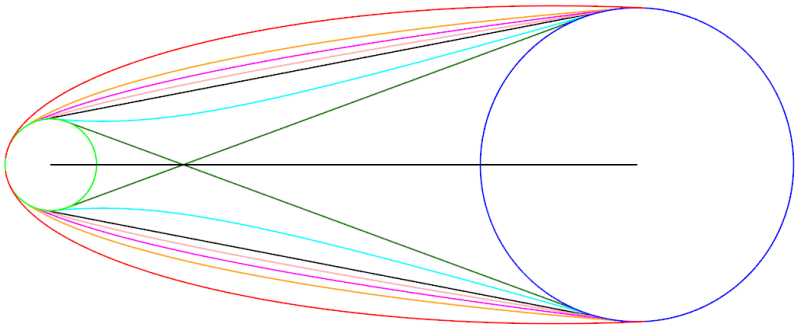


Figure 3.12: Symmetric conics produced by changing parameter a, dark green: valid conic with minimum thickness, cyan: hyperbola opening along y-axis, black: cone, pink: hyperbola opening along x-axis, magenta: parabola, orange: ellipse, red: valid conic with maximum thickness.

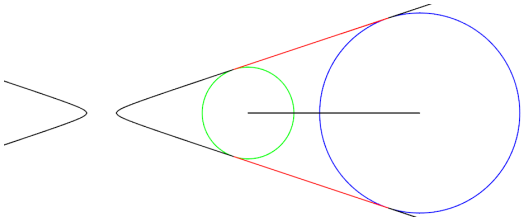


Figure 3.13: Hyperbola with single branch forming the beam.

The parameter a (or $m = a - 1$) provides an algebraic control of the profile curve. This control scheme is not very intuitive. We propose below a new control parameter that has a clear geometric interpretation, and is more intuitive to work with.

3.2.3 Control of a beam's shape by tangency to a mid-circle that can shrink or grow by thickness 't'

We propose to introduce a third circle, $\mathbf{C_3}$, centered at the origin, and we require our symmetric conic to be tangent to it as well (Fig. 3.14). In other words, $\mathbf{C_3}$ is mid-way between the centers of the original two circles, so it is in the "middle" of the beam, in some sense. We denote the radius of this circle by $r + t$, where t is a ***"thickening" parameter***. If $t = 0$, the conic will be a straight line tangent to all three circles, shown as a dashed black curve in Fig. 3.14, and the resulting beam is a cone beam. Negative values of t produce thinner beams, while positive values of t make thicker beams.

The equation of our third circle, $\mathbf{C_3}$, is $x^2+y^2 = (r+t)^2$. Using the same "discriminant $=$

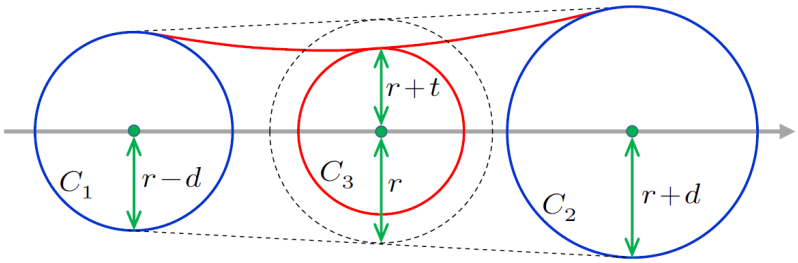


Figure 3.14: Control by mid-circle tangency.

0" technique as in Section 3.2.1, and using expressions for b and c in Eq. 3.7 and Eq. 3.8 respectively, we can show that the conic is tangent to the circle $\mathbf{C_3}$ if

$$m = \frac{1}{d^2 - t^2 - 2rt - 1} \tag{3.11}$$

Since $m = a - 1$, we can compute a in terms of t and use t as the controlling parameter.

$$a(t) = \frac{d^2 - t^2 - 2rt}{d^2 - t^2 - 2rt - 1} \tag{3.12}$$

3.2.4 Valid values of 't' and the corresponding shapes of the beam

The minimum value of t will correspond to the hourglass shape, and can be computed by equating a_{min} (Table 1) to $a(t)$ from Eq. 3.12. Which considering $r + t_{min} > 0$ gives:

$$t_{min} = d - r \tag{3.13}$$

The max value of t will correspond to the elliptical shape when the profile curve loses tangency with the smaller circle. It can be computed by equating a_{max} (Table 1) to $a(t)$ from Eq. 3.12. This gives a quadratic in t, and we pick the larger root:

$$t_{max} = -r + \sqrt{d^2 + (r - d)(r + 1)} \tag{3.14}$$

Fig. 3.15 shows, the symmetric conics produced by varying t in steps of 0.025. The straight line (black) corresponds to $t = 0$. Note that constant increments of t give approxi-

mately constant deformations of the profile curve.

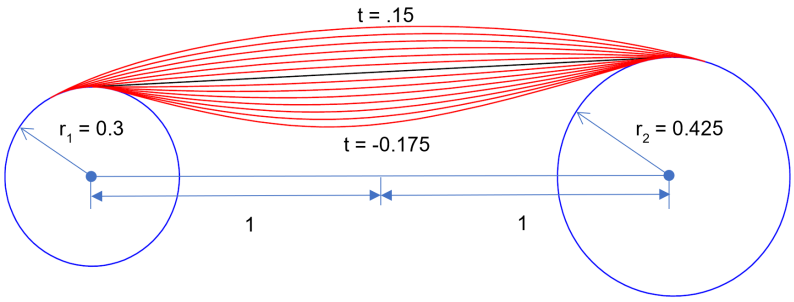


Figure 3.15: Symmetric conics for a range of t values.

3.2.5 Control of a beam's shape by waist thickness, i.e. minimum or maximum thickness of the beam

Observe that neither a, nor t provide a direct control over the min or max cross-section of the beam. We refer to this cross-section as ***waist*** and corresponding beam diameter as waist-thickness w. For example, in Fig. 3.16, observe that even though mid-circle has non-zero radius, the waist-thickness w is close to zero.

We compute waist thickness by writing Eq. 3.1 in the canonical form, i.e.:

$$\frac{(x-(-b/a))^2}{(b/a)^2-c/a}+\frac{y^2}{a((b/a)^2-c/a)}=1 \tag{3.15}$$

We solve for y at $x=-b/a$, and compute waist position and thickness in terms of parameter a:

$$w(a)=2\sqrt{b^2/a-c}\text{ at }(-b/a,0) \tag{3.16}$$

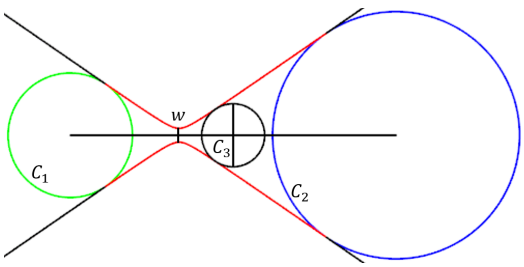


Figure 3.16: Thin waist forming away from origin (center of $\mathbf{C_3}$).

We can then invert Eq. 3.16 to express a as a function of w, but the equations are non-linear. Moreover, the waist can occur outside of the beam (see Fig. 3.17).

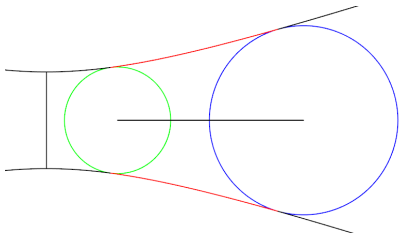


Figure 3.17: Waist forming outside the beam.

We therefore recommend control by mid-circle tangency (Sec. 3.2.3), i.e. by parameter t, for which the equations are relatively simple and the results are intuitive (Fig. 3.15).

3.3 Construction of a biquador beam

Assume circles, $\mathbf{C_1}$ (green) and $\mathbf{C_2}$ (blue) in the plane. Let $\mathbf{A}$ be the axis through their centers. We want a biconic (Sec. 3.1) that is symmetric with respect to $\mathbf{A}$, that is made of two smoothly connected symmetric conic sections and that is tangent to these two circles (see Fig. 3.18).

To relate this solution to the one described in the previous section, we present our solution in terms of a third circle $\mathbf{C_m}$ (black) with its center on $\mathbf{A}$, and that is tangent to the biconic at the junction point X between its two conic parts. Let θ be the angle between the normal $\overrightarrow{N}$ at X and the axis $\mathbf{A}$. One can control the shape of the biquador beam by adjusting the point X and the angle θ.

In Fig. 3.18, we show three vertical lines ($\mathbf{L_1}$, $\mathbf{L_m}$, and $\mathbf{L_2}$) that pass through the tangential control point of the biconic on each circle.

We reduce the computation of a biquador beam to two simpler problems, one on each side of $\mathbf{L_m}$. In each sub-problem, we are given an axis $\mathbf{A}$, a circle $\mathbf{C}$ (standing either for $\mathbf{C_1}$ or for $\mathbf{C_2}$) that has radius r, a point X, and a unit normal $\overrightarrow{N}$. We want a symmetric conic

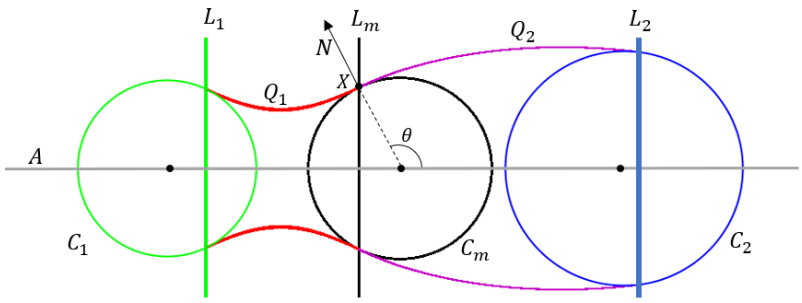


Figure 3.18: Symmetric biconic construction.

$\mathbf{Q}$ that is tangent to $\mathbf{C}$ and that passes through X with normal $\overrightarrow{N}$ at X. This configuration is shown in Fig. 3.19.

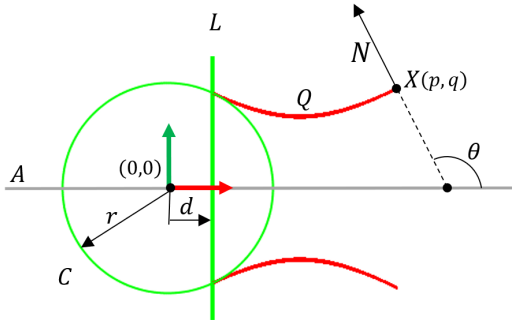


Figure 3.19: Symmetric conic $\mathbf{Q}$ that is tangent to circle $\mathbf{C}$ and passes through point X with normal $\overrightarrow{N}$.

We choose the origin to be the center of $\mathbf{C}$ and axis $\mathbf{A}$ as the x-axis. We compute the equation of conic $\mathbf{Q}$ as follows:

From classical analytic geometry (e.g. Art. 254 on page 212 of [27]), we know that for any given real valued λ the equation

$$x^2 + y^2 - r^2 + \lambda(x - d)^2 = 0 \tag{3.17}$$

represents a conic $\mathbf{Q}$ that is tangent to the circle $\mathbf{C}$ at the two points where it intersects the vertical line $x = d$. We solve for λ and d as follows:

Conic $\mathbf{Q}$ passes through the given point $X = (p, q)$ if:

$$p^2 + q^2 - r^2 + \lambda(p - d)^2 = 0 \tag{3.18}$$

We then differentiate Eq. 3.17 to compute the slope s of the normal to the conic $\mathbf{Q}$. At X, $s = \tan\theta$, which gives:

$$s = \frac{q}{d - p(1 + \lambda)} \tag{3.19}$$

We then solve the system of equations (3.18) and (3.19) to obtain:

$$d = \frac{sq^2 - sr^2 + pq}{q - sp} \quad ; \quad \lambda = \frac{q - sp}{s(p - d)}$$

Fig. 3.20 shows examples of symmetric biconics. Fig. 3.21 shows examples of biquador beams constructed using symmetric biconics. Biquador beams make it easier to use waist thickness as a way to control the shape of the beam. For this, we fix $\theta = 90°$. Results are shown in Fig. 3.22.

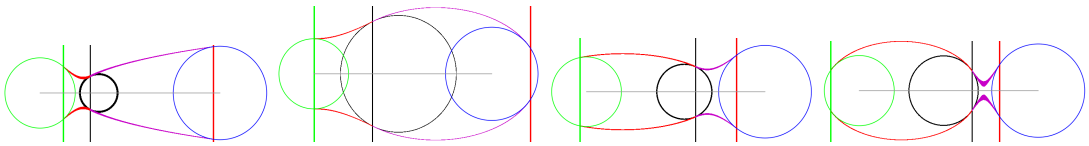


Figure 3.20: Symmetric biconics of the same two circles.

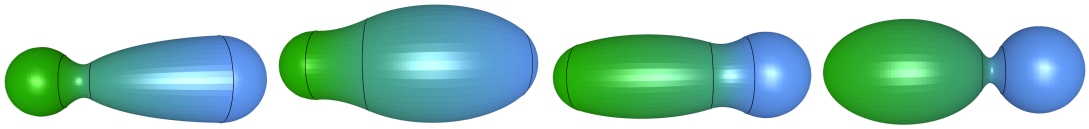


Figure 3.21: Biquador beams of the same two balls.

3.4 Experimental validation of our approach to construct quador beams

We have tested our approach (Sec. 3.2.3) to construct a quador beam on a pair of balls. In our tests, we vary the radii of the balls and distance between them. For fixed values of

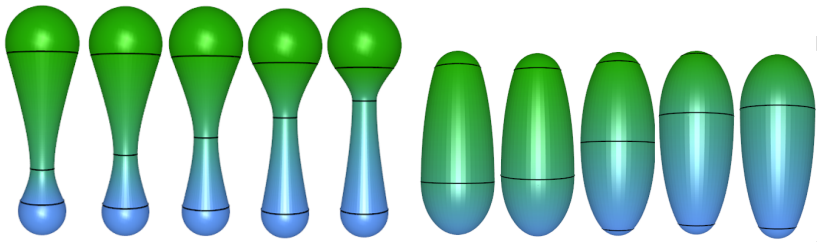


Figure 3.22: Examples of biquador beams between identical pairs of balls. The beams have different shapes but same minimum or maximum waist thickness.

radii and distance between the two balls, we incrementally vary the thickening parameter t from its minimum value (Eq. 3.13) corresponding to an hourglass shape (Fig. 3.12) to its maximum value (Eq. 3.14) corresponding to an ellipsoid shape (Fig. 3.12). Fig. 3.15 in Sec.3.2.4 shows profile curves corresponding to a subset of valid values of t. Outside of the valid range for t, we obtain invalid profile curves, such as those shown in Fig. 3.11. Similarly, we tested our biquador beam construction (Sec. 3.3) for different pairs of balls. For each pair, we gradually vary the three control parameters (coordinates of point X and angle θ) to create a variety of biquador beams.

3.5 Engineering benefits of quador beams

In previous sections, we have shown that the computational complexity of quador beams is no worse than cylinder or cone beams. However, from an engineering perspective, lattices constructed from quador beams have some significant advantages, which are described in this section.

In order to increase their strength-to-weight ratio, microlattices are often optimized by varying the diameters of their nodes and beams.If a lattice has cylindrical beams, then the tangency condition that we have used means that all beams and all nodes will have the same diameter, so there is only a single variable that can be used in editing and optimization. Freedom can be increased by relaxing the tangency condition, but then beam/beam intersections become much more complex, and it is impossible to derive any computational benefit from the properties discussed in this thesis.

For lattices with cone beams and with tangent node/beam junctions, thickness gradation can be achieved by changing the diameters of the nodes. This implicitly fixes the diameters of the beams. With this approach, it is impossible to make beam thickness dependent on the beam orientation, which restricts opportunities for optimization.

In lattices with quador or biquador beams, beam diameters can be controlled independently of node diameters. For example, with quador beams, the thickening parameter t discussed in Sec. 3.2.3 can be used as one of the design variables. The increased freedom does not adversely affect the complexity of the geometry at lattice nodes, but it provides a larger design space for optimization, so better results can be achieved. For example, if a lattice is supporting a load, it is quite likely that an optimal design will have thin horizontal beams and thicker vertical ones. This is possible with quador beams, as shown in Fig. 3.23, but not with conical ones.

When optimizing a lattice of quador beams, all of the individual node and beam diameters are potentially available for use as independent variables. However, a complex lattice may have billions of nodes and beams, so using individual diameters would give us far more variables than a typical optimization process could handle. A better approach is to use field functions to define node and beam diameters. So, for example, we might have two real-valued functions f_N and f_B that give us the node and beam diameters at any point in the lattice in terms of a few dozen design parameters. These parameters would be used as the independent variables in optimization processes, not the individual node and beam diameters.

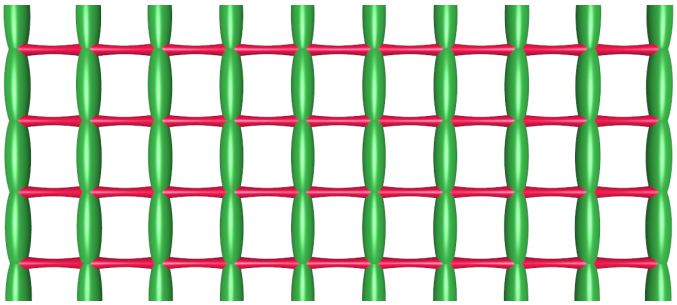
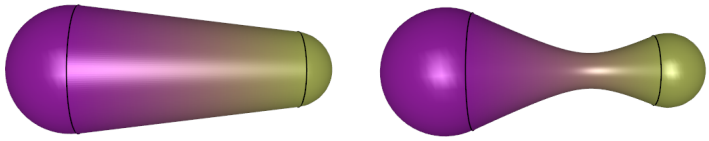


Figure 3.23: Varying beam diameters



(a) area/volume ≈ 0.073 (b) area/volume ≈ 0.110

Figure 3.24: Area/volume ratio

Another advantage of using quador or biquador beams over cylinder or cone beams is that one can significantly vary the ratio of their surface area to volume. For example, the area/volume ratio of the quador beam in Fig. 3.24 is roughly $1.5\times$ that of the cone beam, even though their nodes are the same size. This is important when designing lattices for applications where surface area is important, such as those involving convection, and adsorption. Hence, we expect that profiled beams may have benefits for structural performance [28], or heat transfer [29].

Microlattices are also optimized by designing them as hierarchical or multilevel structures [30], essentially replacing each ball or beam of the lattice with a lattice and so on. We expect that quador and biquador beams may be useful at the coarsest level, i.e. level 1 to support overall aesthetics or functional design goals, at level 2 to support optimization that adjusts the thickness of individual beams while ensuring that beams not incident on the same ball don't intersect, and at the level 3 to help to redistribute mass.

To summarize, we described in this chapter two new families of beams, namely the quador and the biquador beams, and discussed simple and intuitive schemes to construct them and to control their shapes. These beams are bounded by quadric-of-revolution surfaces. Two such surfaces in general may intersect in a degree 4 curve, which is complex to work with. In the next chapter, we discuss special properties of quador (and biquador) beams that reduces this complexity, so that we may compute the intersections of surfaces in a lattice of quador beams reliably and accurately.

CHAPTER 4

LATTICE AS AN ASSEMBLY OF HUBS

In Ch. 1 we defined a lattice as an assembly of balls and beams, wherein each beam connects two balls. For structural integrity most beams in such a lattice are kept disjoint by design, but not the beams that are incident on the same ball, in fact intersections of beams incident on the same ball may provide additional strength to the lattice junction at that ball. In this chapter we first discuss a new decomposition of a lattice. In this decomposition, instead of balls and beams as constituent units, we consider each ball and a part of each beam incident on that ball as a new unit. We define this unit in the next section. Thereafter, we discuss special properties of this unit, that makes this decomposition particularly attractive for computing a precise representation of the lattice.

4.1 Definition of a hub and of a clean lattice as an assembly of hubs

Inspired by [12], we split each beam of a lattice into two ***half-beams*** by a plane normal to that beam's axis. For a quador beam, we place the dividing plane at equal distances from the surface of the two balls of the beam (Fig. 4.1). A biquador beam is divided at the junction between its two quador surfaces. Splitting a beam into half-beams introduces a ***cap*** face in the shape of a disk at the end of each half-beam (Fig. 4.1). We then define a ***hub*** as the union of a node (ball) and the ***half-beams*** incident on that node (Fig. 4.1).

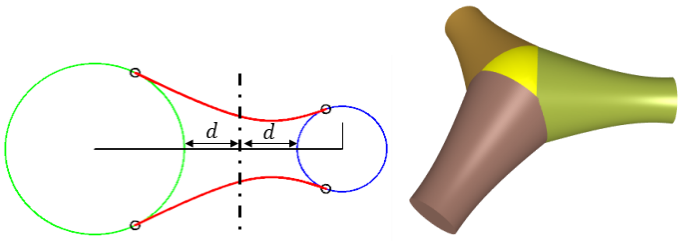


Figure 4.1: Splitting of a beam into two half-beams (left). A hub with 3 half-beams (right).

We say that the lattice is ***clean***, when its hubs are quasi-disjoint, i.e., when their interiors are disjoint and only the half-beams of the same beam are in contact at their common cap disk. Typically, architected lattices are intentionally clean, so as to facilitate analysis and optimization. But programmed gradations or warps of regular lattices may produce unclean results. The left subfigure in Fig. 4.2 shows a lattice represented as an assembly of balls and beams, and in the right subfigure the same lattice is shown as an assembly of hubs.

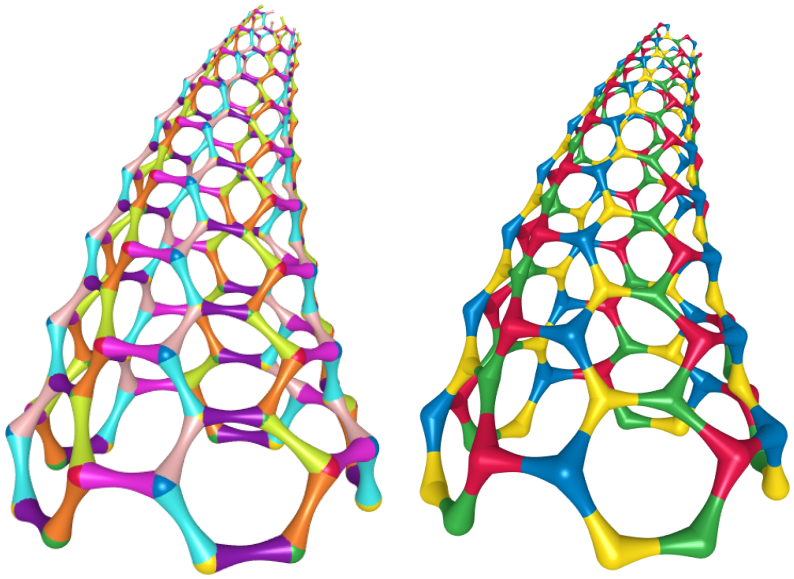


Figure 4.2: A lattice shown as the union of its balls and beams (left). The same lattice shown as an assembly of hubs (right). Individual hubs are shown in different colors .

In the next section, we discuss special properties of a hub, that makes makes it convenient to capture the geometry and topology of a hub.

4.2 Special property of a hub - intersection curves of two quador surfaces of a hub are conic sections and hence planar

The intersection of two general quadric surfaces is a space curve of degree 4 (Fig. 4.3-left). Its computation and processing have received much attention in the CAD and CAGD communities [31].

However, the intersection curve of any two quador surfaces of a hub is either empty or

conic sections, and hence planar (Fig. 4.3-right).

This fact follows from Salmon's observation in Art. 140 of [32], on page 137, where he states that *"Two quadrics having plane contact with the same third quadric intersect each other in plane curves."*.

In the case of a hub, the first two quadrics are the curved faces of the half-beams and the third quadric is the surface of the sphere of the hub. For the special case of two cones having a common inscribed sphere, the planar intersection curves were studied by Shene in [33]. Our discussion here covers the more general case of *any* two intersecting quador half-beams of a hub.

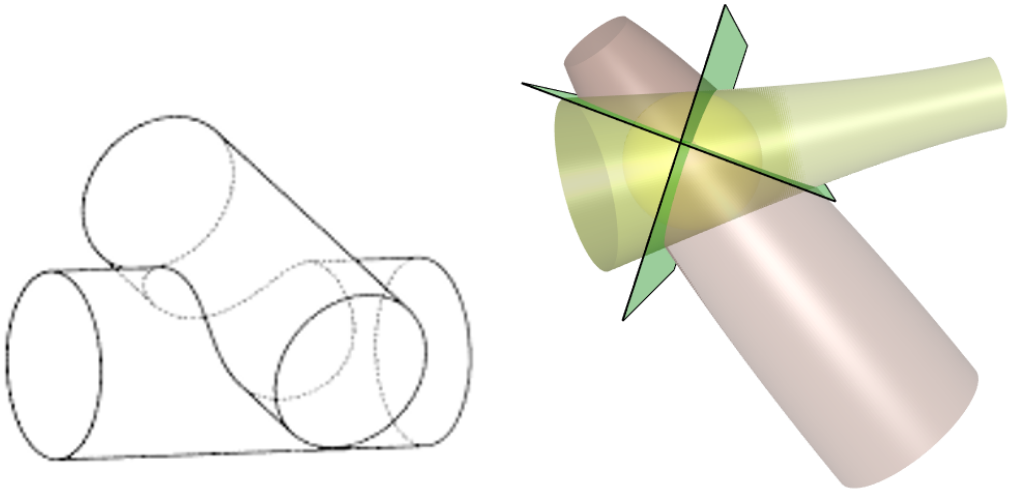


Figure 4.3: The intersection of two quadric surfaces in general is a curve of degree 4 (left) (image taken from [34]), while two quador surfaces in a hub intersect in conic (degree 2) curves that lie in two different planes (right).

4.2.1 Computing the planes of intersection of two quador surfaces of a hub

We follow the common convention of using the same symbol to denote a polynomial expression and its zero set. So, for example, the symbol $\mathbf{S}$ might denote both the expression $\mathbf{S}(x, y, z) = x^2 + y^2 + z^2 - 1$, and the spherical surface $\{(x, y, z) \in \mathbf{R}^3 : \mathbf{S}(x, y, z) = 0\}$. Strictly speaking, this is an abuse of language and notation, but it is commonplace and convenient.

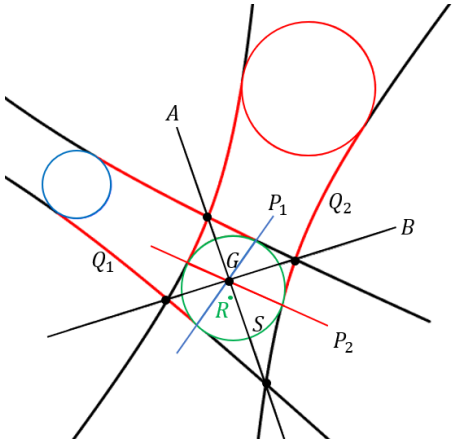


Figure 4.4: Cross-section of two intersecting quadric surfaces with common inscribed sphere, by a plane that contains their axes.

Consider the quadric surfaces Q_1 and Q_2 corresponding to two half-beams of a given hub, with the ball $\mathbf{S}$ as their common inscribed sphere (Fig. 4.4). For $i = 1, 2$, let P_i be the ***contact-plane*** containing the intersection of $\mathbf{Q_i}$ and $\mathbf{S}$. There exist real numbers λ_1 and λ_2 such that $Q_1 = \mathbf{S} + \lambda_1 {\mathbf{P_1}}^2$ and $Q_2 = \mathbf{S} + \lambda_2 {\mathbf{P_2}}^2$. Where Q_1 and Q_2 intersect, we have $Q_1 = Q_2$, i.e., $\mathbf{S} + \lambda_1 {\mathbf{P_1}}^2 = \mathbf{S} + \lambda_2 {\mathbf{P_2}}^2$, which reduces to $\lambda_1 {\mathbf{P_1}}^2 - \lambda_2 {\mathbf{P_2}}^2 = 0$. Upon factorizing we obtain:

$$(\sqrt{|\lambda_1|}\mathbf{P_1} - \sqrt{|\lambda_2|}\mathbf{P_2})(\sqrt{|\lambda_1|}\mathbf{P_1} + \sqrt{|\lambda_2|}\mathbf{P_2}) = 0$$

In other words, the intersection of Q_1 and Q_2 is contained within the union of the two planes

$$\begin{aligned} \mathbf{A} &= \sqrt{|\lambda_1|}\mathbf{P_1} - \sqrt{|\lambda_2|}\mathbf{P_2} \\ \mathbf{B} &= \sqrt{|\lambda_1|}\mathbf{P_1} + \sqrt{|\lambda_2|}\mathbf{P_2} \end{aligned} \tag{4.1}$$

If $\lambda_i = 0$, then we have $\mathbf{Q_i} = \mathbf{S}$, and such cases are of no interest to us.

Let R be the center of sphere $\mathbf{S}$. Using the techniques described in Sec. 3.3, we can find the distance d from R to the plane $\mathbf{P_1}$. Then, if U is the unit vector along the axis of Q_1 away from R, point $R + dU$ lies on the plane $\mathbf{P_1}$. So, the plane $\mathbf{P_1}$ consists of those points X satisfying the equation $RX \cdot U = d$. Once $\mathbf{P_1}$ is known, we can equate coefficients in the

identity $Q_1 = \mathbf{S} + \lambda_1 \mathbf{P_1}^2$ to calculate λ_1. The plane $\mathbf{P_2}$ and the scalar λ_2 can be obtained by similar reasoning, and then Eq. 4.1 gives us the desired planes $\mathbf{A}$ and $\mathbf{B}$ containing the two intersection curves of the entire quador surfaces.

We now prove that the intersection curve of two half-beams corresponding to surfaces $\mathbf{Q_1}$ and $\mathbf{Q_2}$ is contained in plane $\mathbf{A}$ and not in the plane $\mathbf{B}$ of Equation (4.1). Consider the configurations of the two contact-planes $\mathbf{P_1}$ and $\mathbf{P_2}$ in Fig. 4.5. We can orient the normals of the planes $\mathbf{P_1}$ and $\mathbf{P_2}$ so that each half-space $\mathbf{H_i} = \{X : \mathbf{P_i}(X) \geq 0\}$ contains the corresponding half-beam. Then the intersection curve, $\mathbf{C}$, lies in the wedge-shaped region $\mathbf{H_1} \cap \mathbf{H_2}$. Following generalized Brianchon Throrem (Appendix. B), $\mathbf{P_1}$, $\mathbf{P_2}$, $\mathbf{A}$ and $\mathbf{B}$ are concurrent at G. Now, if we take a point X in the interior of $\mathbf{C}$, then $\mathbf{P_1}(X) > 0$ and $\mathbf{P_2}(X) > 0$. By definition, we have $\mathbf{B}(X) = \sqrt{|\lambda_1|}\mathbf{P_1}(X) + \sqrt{|\lambda_2|}\mathbf{P_2}(X)$, as $\sqrt{|\lambda_1|} > 0$ and $\sqrt{|\lambda_2|} > 0$, so $\mathbf{B}(X) > 0$, and hence $X \notin \mathbf{B}$. This shows that $\mathbf{C}$ is not contained in $\mathbf{B}$, so it must be contained in $\mathbf{A}$. We orient the normal to plane $\mathbf{A}$, such that the corresponding half-space $\mathbf{A_{ij}}$ contains the center of the other ball of the quador beam corresponding to half-beam $\mathbf{S_i}$. In fact, the normal defined by Equation (4.1) for plane $\mathbf{A}$ provides its correct orientation. Fig. 4.6 shows that the intersection curve of two half-beams of a hub lie in a plane.

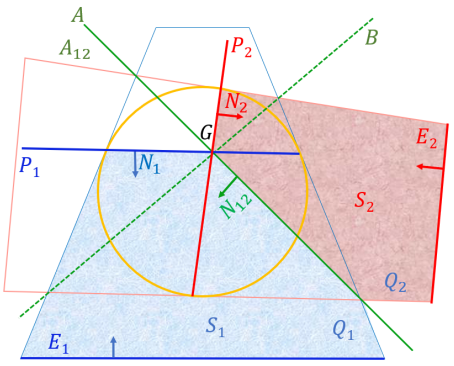


Figure 4.5: Cross-section of a hub with two half-beams, $\mathbf{S_1}$ and $\mathbf{S_2}$. Their trimmed portions (shaded regions). The contact planes $\mathbf{P_1}$ and $\mathbf{P_2}$, and the corresponding half-spaces, indicated by the inward normal vector. Plane $\mathbf{A}$ and $\mathbf{B}$ are the Intersection-planes. Half-space $\mathbf{A_{12}}$ is oriented to contain the the trimmed version of $\mathbf{S_1}$.

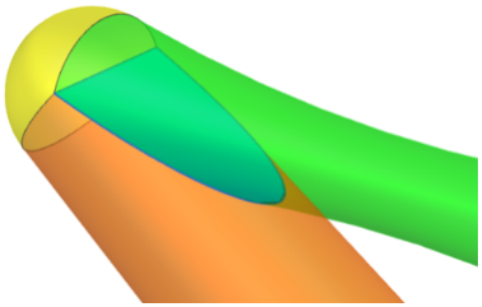


Figure 4.6: The intersection curve of two half-beams of a hub lie in a plane.

4.2.2 Computing the plane-quador intersection curves

In Sec. 4.2.1, we saw that the intersection of two quador surfaces of the same hub can be computed by intersecting either of them with a plane. So, in this section, we turn to the problem of intersecting a quador surface $\mathbf{Q}$ with a plane $\mathbf{A}$.

The intersection may be computed using general techniques that apply to any surface of revolution [35]. However, the process may be simplified by taking advantage of the fact that the surface is a quadric-of-revolution [36]. In most cases that occur in lattice geometry, the intersection curve is an ellipse, $\mathbf{E}$, (Fig. 4.7), which may be calculated as follows.

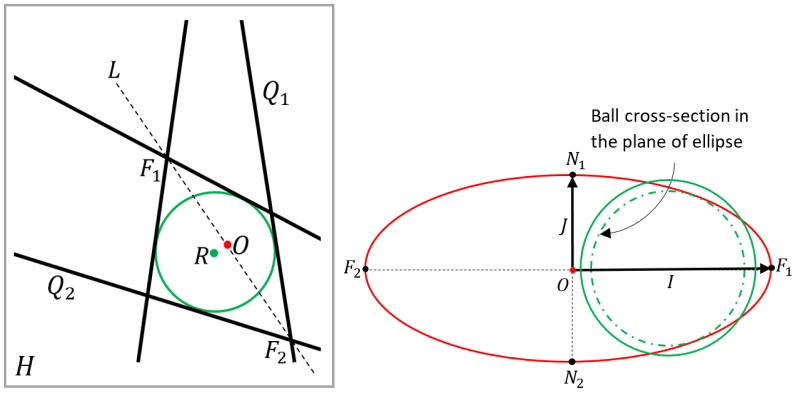


Figure 4.7: Computing the intersection curve between two quador surfaces.

Consider the plane $\mathbf{H}$ that contains the axis of the quador surface $\mathbf{Q}$ and is normal to the plane $\mathbf{A}$. Let $\mathbf{L}$ be the line of intersection of this plane and the plane $\mathbf{A}$. We intersect $\mathbf{L}$ with the quadric $\mathbf{Q}$ to obtain two points F_1 and F_2. These are the end-points of the longer axis of the ellipse $\mathbf{E}$. The center of the ellipse is therefore the point $O = \frac{1}{2}(F_1 + F_2)$. Next, we

intersect $\mathbf{Q}$ with a line through O that is normal to the plane $\mathbf{H}$. The resulting intersection points N_1 and N_2 are the end-points of the shorter axis of the ellipse. If $\overrightarrow{I} = \overrightarrow{OF_1}$ and $\overrightarrow{J} = \overrightarrow{ON_1}$, then the ellipse has the parametric equation:

$$E(t) = O + \cos(t)\overrightarrow{I} + \sin(t)\overrightarrow{J}$$

4.3 Experiments to test our approach to compute the intersection-plane

In this section, we discuss a few experiments that we performed to test if the our approach to compute the intersection-plane is giving correct results. In Sec. 4.2.1, we proved that the intersection between two quador surfaces of a hub is planar and that the intersection curve of the two half-beams of a hub lies in the plane $\mathbf{A}$ obtained by Eq. 4.1. We perform experiments to test if the plane $\mathbf{A}$ obtained by Eq. 4.1 is indeed the intersection-plane of the surfaces of two half-beams of a hub as follows. We instantiate a hub with two quador half-beams. We color the two half-beams with different colors. We compute the intersection-plane $\mathbf{A}$ of the surfaces of the two half-beams using the technique described in Sec. 4.2.1. Keeping one of the beams stationary, we gradually move the other beam around the stationary beam so as to change the plane of intersection between them. We visually inspect to ensure that the surface of each half-beam is visible only on one side of the intersection-plane (Fig. 4.8). We also vary the profiles of the two half-beams to perform this test for a variety of quador beams.

In this chapter we proposed a new decomposition of a lattice. In subsequent chapters, we exploit this decomposition in different ways. In the next chapter, Ch. 5, we propose three representations to capture the geometry and topology of a hub, which for a clean lattice (disjoint hubs) can be extended trivially to the whole lattice. Later, in Ch. 7, we propose a class of lattices wherein a lattice consists of rows (sequence) of hubs, such that the hubs of a row are related to the first hub of that row by similarity transforms. Thus the representations of all hubs in the lattice may be computed from the representations of one

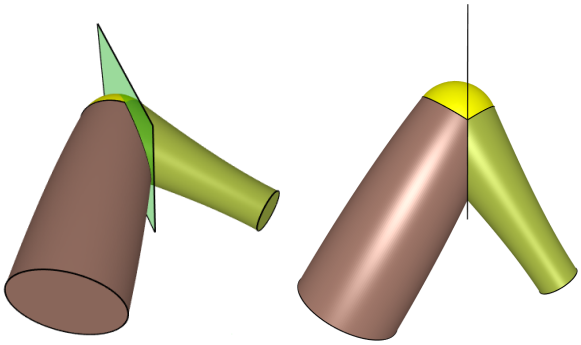


Figure 4.8: A hub with two half-beams and the plane of intersection of the surfaces of the two half-beams (left). Top view of the hub and the plane of the left subfigure, showing that the regions belonging exclusively to each of the two beams are separated by this intersection-plane (right).

hub per row. Finally in Ch. 9, we discuss distributing (for parallel processing) a geometric query on the lattice to its hubs, and propose efficient algorithms to perform geometric queries on a hub.

CHAPTER 5

REPRESENTATIONS OF LATTICES WITH QUADOR BEAMS

To support lattices with quador beams, it is important to be able to perform queries on them reliably, and for that it is important to capture their geometry and topology, accurately and precisely. In this chapter, we discuss computing three ***representations*** of a hub, namely the ***Constructive Solid Geometry (CSG)*** [37] representation, the ***Constructive Solid Trimming (CST)*** representation [38], and the ***Boundary Representation (Brep)*** [37]) (Fig. 5.1). In a clean lattice (Sec. 4.1), the interiors of the hubs are disjoint and the closure of two hubs that are connected by a beam intersect in a disk. This observation makes it trivial to remove these disks and to stitch the representations of individual hubs into a single representation for the lattice. We capture the analytical representation of the underlying curves and surfaces instead of their polygonal approximations.

The three representations of a hub have special properties that makes each of them suitable for specific queries over the other two representations. For example, the CSG representation is defined using quador and linear half-spaces, and is more suitable for ***Point Membership Classification (PMC)*** than CST and Brep. The CST representation uses quador half-spaces, each trimmed by a ***Convex Trimming Polyhedron (CTP)***, and is useful in volume meshing of a hub. The Brep has only conic curves and quador surfaces, and is convenient to perform minimum distance queries to the boundary of a hub. Furthermore, computing conic curves of the Brep requires intersecting a plane with a quador surface, which is much simpler than computing the intersection between two quadric surfaces [39, 40]. Thus surface-to-surface intersections can be easily modeled without "cracks" [37, 41]. Also, compared to spline approximations or procedural curves [42, 43], conics require less storage space, and allow faster and more reliable geometric computations. We discuss geometric queries on a hub in detail in Ch. 9.

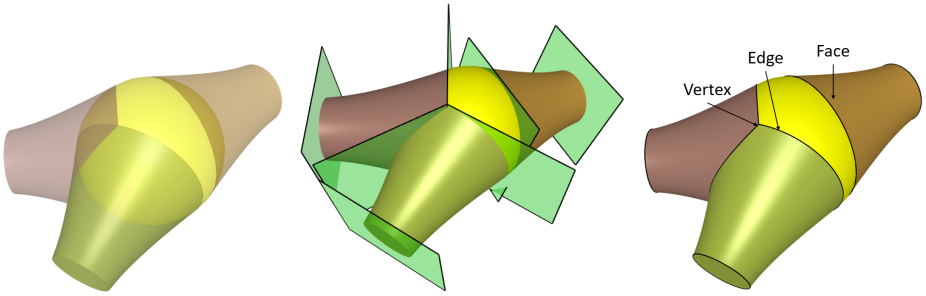


Figure 5.1: Representations of a hub: CSG (left), CST (middle) Brep (right)

The chapter is organised as follows. In Sec. 5.1, we discuss the prior art on computing representations of a solid. Then, in Sec. 5.2, we propose the three (CSG, CST and Brep) representations of a hub, and in Sec. 5.3, we describe a compact and efficient data structure to store and operate on these representations. In Sec. 5.4, we describe how we compute these representations. In Sec. 5.5, we present several functions that operate on the proposed data structure. These operators can be used to query the geometry or the topological relationship of different elements of the three representations.

5.1 Prior art on representations of a solid

Several references [44, 41, 45, 42] discuss computing the Brep of a solid from its CSG representation. Our interest is particularly on the computation of the exact Brep (using underlying curves and surfaces not their polygonal approximations) of a solid from its CSG representation. There are two types of challenges that the research on computing the exact Brep has focused on: (1) the challenges associated with the conversion from one representation to the other, in this case from CSG to Brep [46, 47, 48], and (2) the challenges associated with the accuracy of numerical arithmetic for computing both polyhedral [49] and exact boundary representations [50, 51].

As for the kind of solids, some research focuses on computing the Brep of polyhedral solids [52, 53], while others consider solids with curved surfaces [39, 50]. One of the major challenges in computing the exact Brep of solids with curved surfaces [50], is

the computation of surface/surface intersection curves and vertices. In many cases it is not feasible to represent these curves exactly, so they are often approximated by spline curves. It is quite common [42, 43] to store three separate spline representations: one 3D spline, and one spline expressed in the parameter space of each adjacent surface. These three spline curves are not geometrically identical, which often leads to "cracks" along the model's edges. Properly designed applications algorithms may use topological information to avoid problems with the "cracks", but elaborate tolerance schemes are required [54]. Another approach is to represent the intersection curve procedurally, whereby any desired point on the curve is calculated by intersecting the two adjacent surfaces with some third surface (usually a plane). Regardless of which approach is used (spline approximation or procedural curves), large data size and slow computation speeds are unresolved issues. Simple analytic curves require far less space and less computation time, and lead to more reliable applications.

In a way, for a hub, the surface-surface intersection problem is simpler as it involves only quadric surfaces. Early on, [39] proposed exact intersection between quadric solids in the context of the PADL-2 geometric modeling system, but it only dealt with "natural" primitives, namely (planes, spheres, cylinders and cones) and defined curve types as intersection between specific primitives, such as "cyl-cyl", or "sph-cyl". For a hub, each surface is trimmed by a CTP (Convex Trimming Polyhedron) (possibly unbounded), hence we compute an intersection curve by intersecting a plane with a quadric surface, which is much simpler than intersecting two quadric surfaces.

More recently, several authors have proposed computing exact intersections between quadrics [55, 56, 57, 58]. In the special case of a hub, we reduce all topological decisions (for computing the three representations of the hub), to checking the sign of expressions that involve only the standard arithmetic operators $(,*,-,/)$ and the square root. So, our approach requires operating only on numbers of the form $a + \sqrt{b}$, where a and b are extended-precision rationals.

In a hub, all intersection curves are conics and have closed-form expressions, which allows us to model surface-to-surface intersections without "cracks". Also, compared to spline approximations or procedural curves, conics require less storage space, and allow faster and more reliable geometric computations. Furthermore, a vertex of the hub is the intersection of three conics. We use these special properties to build a compact Brep, by adapting the corner table [13] data structure.

Different kinds of geometric queries [44, 41, 45, 42] can be performed on a solid, such as Point Membership Classification (PMC) [59], ray intersection [60], and cross-sectioning [61]. These fundamental queries may be used to make more complex queries, such as computation of mass properties [10, 62] or to perform complex operations on the solid, such as offsetting [63]. The efficiency of a query depends on the representation of the solid; for example, performing PMC is easier on a CSG representation than a Brep, and offsetting operations are easier to perform on the Brep.

To summarize, the Brep of the solids with curved surfaces, may have "cracks" along their edges. A limited set of solids can be represented exactly, but it requires an elaborate system to perform exact arithmetic computations. We propose representations of a hub — the building block of a lattice with quador-beams. Our approach reduces topological decisions to simple and robust computations. We provide algorithms for efficient querying of the hub, a key requirement to support large and complex lattices.

5.2 Representations of a hub

Consider the hub shown in Fig. 5.2. It shows the three kinds of planes. The ***contact-planes*** of the surfaces of quador beams with that of the ball, the ***split-planes***, each splitting a beam into two half-beams, and the ***intersection plane*** between a pair of half-beams.

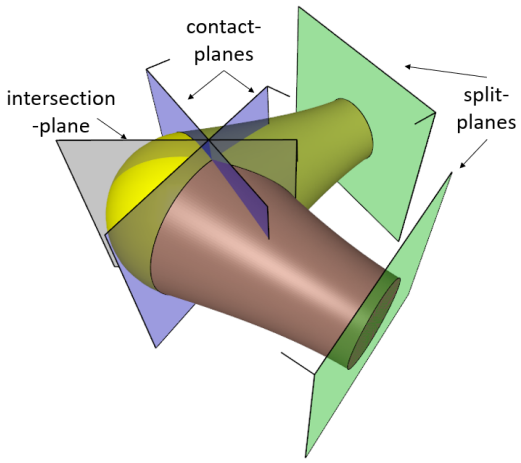


Figure 5.2: Trimming planes in a hub

5.2.1 CSG representation of a hub

We start with a "natural" CSG representation of a hub, which is the union of the following solid primitives: a solid ball and a number of half-beams, each defined by the intersection of a quadric-of-revolution (quador) half-space with two linear half-spaces, one half-space bounded by the *contact plane* and the other half-space by the *split plane* of the beam. Both planes are orthogonal to the axis of the corresponding beam. Fig. 5.3, shows a hub as the union of ball $\mathbf{B_0}$ and three half-beams $\mathbf{H_1}, \mathbf{H_2}$ and $\mathbf{H_3}$. It also shows a half-beam $\mathbf{H}$ as the intersection of the negative half-space of quador surface $\mathbf{Q}$ and positive half-spaces of the contact-plane $\mathbf{L_1}$ and the split-plane $\mathbf{L_2}$.

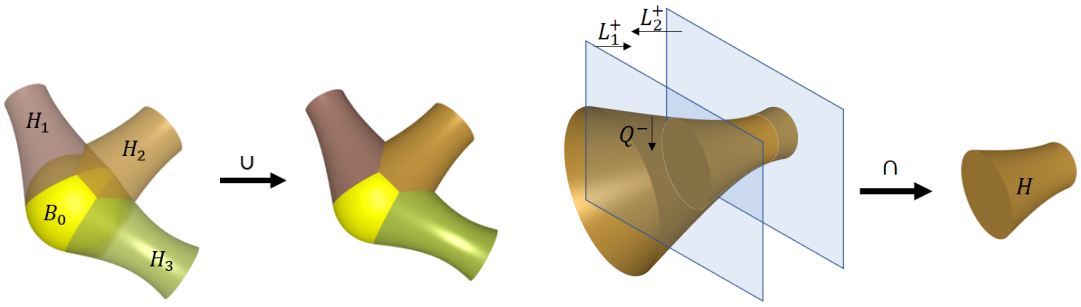


Figure 5.3: CSG representation of a hub(left) and CSG representation of a quador half-beam (right).

5.2.2 CST (Constructive Solid Trimming) representation of a hub

The natural CSG representation of a hub (as the union of the ball and the quador half-beams), defines a "natural" Constructive Solid Trimming (CST) expression [38] for each face of the hub. For example, the spherical face of the hub is the portion of the ball's surface that is outside of the union of the half-beams. The definition of the CST for each face is based on the active zone [46] of the corresponding primitive in the natural CSG expression of the hub.

While, the natural CST of a hub is useful in doing some queries, such as classifying candidate points on the surface of the hub, e.g. in ray intersections against active zones [46], it requires trimming a quadric surface by a set of quadric half-spaces. Since, the intersection of surfaces of a hub is planar, we propose a CST variant that only requires trimming a quador surface by a Convex Trimming Polyhedron (CTP) [1]. The CTP of the ball (Fig. 5.4) is the intersection of the planar half-spaces that are each bounded by a contact-plane. The CTP of a beam (Fig. 5.4) is the intersection of the following planar half-spaces: those two that are bounded by the contact- and by the split-planes and also the *intersection-planes* of that beam with all the other beams.

The CTP based CST may be useful in supporting several geometric queries. For example, in segmenting the interior of the hub (into disjoint regions separated by planes) for volume meshing or classifying a set of candidate points on a surface against corresponding CTP or in trimming parametric curves defined on a surface.

5.2.3 Boundary representation (Brep) of a hub

A hub is the region bounded by a set of *faces* $\{f_i\}$ bound by a set of *edges* $\{e_i\}$ meeting in a set of *vertices* $\{v_i\}$. We represent the elements (surfaces, faces, loops, intersection curves, edges, vertices) (Fig. 5.5) of the boundary of the hub as follows:

- Surface: Each of the half-beams of a hub is bounded by a quadric-of-revolution

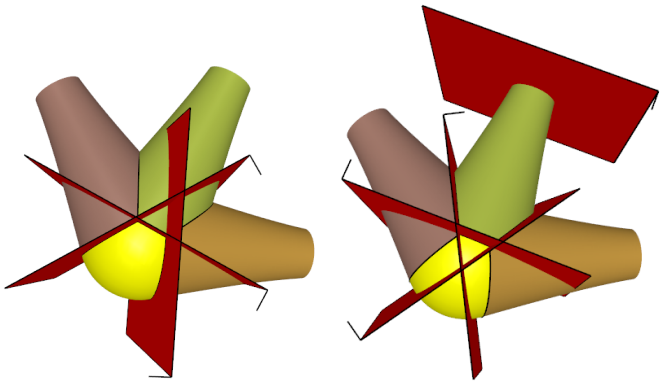


Figure 5.4: CST representation of a hub. Oriented planes of the trimming polyhedron of the yellow ball (left) and of the green half-beam (right).

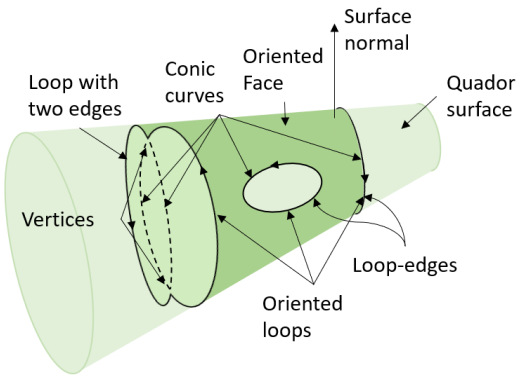


Figure 5.5: Illustration of Brep elements on a quador half-beam.

(quador) surface. Following [1], we represent a quador surface by a conic profile in a plane though the axis of the beam. We define the plane, by a normal and a coordinate system with its x-axis as the axis-of-revolution. Each surface is oriented to have an outward surface normal (away from primitive's center/axis). We represent the spherical surface of the hub's ball as a quador surface, also.

- Face: Each face of the hub (including the faces on the hub's ball) is a portion of a quador surface that is bounded by a set of oriented *loops* (Fig. 5.5). A surface may contribute multiple faces, and each face may have multiple loops.

- Loop: A *loop* is either a single ellipse or a cycle of edges, each being a conic curve. Each loop is oriented such that when walking along it, the face it bounds lies on

its right. Note that some of these loops don't have a vertex, we refer to them as ***loop-edges*** (Fig. 5.5).

- Intersection curve: Each of the intersection curves of a hub is a conic defined by the intersection of a contact-, split-, or intersection-plane with a quador surface. While it is apparent that the contact- or split-plane of a beam is defined by the position of the hub's ball and the details of the beam, it's noteworthy that the intersection-plane between two beams is defined implicitly from the contact-planes of the two beams. This property is crucial to consistently parameterize each intersection curve.

- Edge: Each edge is a conic segment (or an ellipse in case it is a loop-edge) defined by parameter values (t_{start}, t_{end}) on an underlying parameterized intersection curve.

- Vertex: We assume ***general position***, i.e. no four surfaces pass though the same point. With this assumption, each vertex of the hub lies on the intersection of three surfaces (Fig. 5.6), two of which are of half-beams and the third is either of the ball or of another half-beam. Each pair of surfaces in a triplet of surfaces intersect in a plane. Remarkably, these three planes always intersect in a line [1], and the intersection of that line with any one of the three surfaces gives two vertices. We differentiate between these two vertices using the clockwise ordering of surfaces around each vertex, with respect to the outward direction at that vertex along the line (Fig. 5.6 right). We describe in Sec. 5.4.3, how we consistently compute this ordering at each vertex

5.3 Compact data structure to build the three representations of a hub

Our goal is to build a compact data structure for the three representations of a hub, that allows efficient querying of the geometry and topology of the hub. We adapt the Corner-Table data structure [13], to accommodate curved faces bounded by loops, each loop having single or multiple curved edges. A corner represents the association of a face with a vertex.

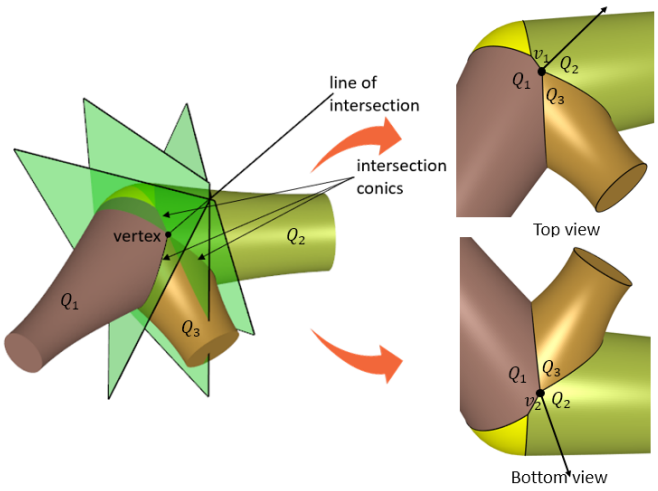


Figure 5.6: An example showing two vertices of a hub on the intersection of three quador surfaces indexed as $(1, 2, 3)$, which produces two vertices (v_1, v_2), that lie on the intersection of a line with any one of the three surfaces. We differentiate between the two vertices using clockwise ordering of surfaces, i.e., $(1, 2, 3)$ at v_1 and $(1, 3, 2)$ at v_2, with respect to outward direction along the line at each vertex.

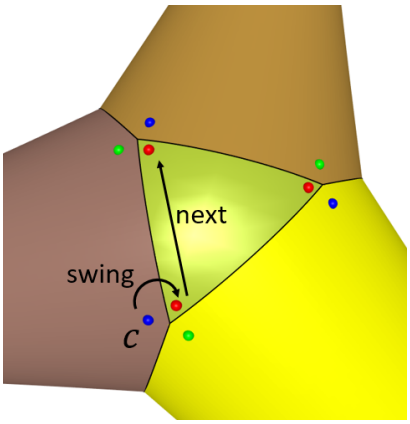


Figure 5.7: Corner-Table representation of the Brep of a hub: each of the three faces incident on a vertex contributes a corner. These are shown as colored dots around each vertex. We can "swing" around a vertex to go from one of its corner to its corner in the neighbouring face, or go to the "next" corner along a loop in the face.

In designing this data structure, we take advantage of several special properties of the hub. For example:

1. Each vertex of the hub is the intersection of three quador surfaces, therefore we consider the boundary of the hub to be, what we call, a $\mathbf{Y}$-mesh. Furthermore, as each vertex corresponds to 3 corners, we don't store explicit information to swing

from one face to another around a vertex (see Fig. 5.7), but instead assign corner IDs so that the three corners of a vertex have consecutive IDs.

2. In general two quadric surfaces intersect along a non-planar curve [64, 65], which may degenerate into two conics (planar curves) [32]. In case of a hub, the intersection of the quador surfaces of two half-beams, is contained in a single conic [1] (Fig. 5.6).

3. In general, three quadric surfaces may intersect in at most 8 points [66], but for a hub, three quador surfaces may intersect in at most two vertices (Fig. 5.6).

4. We identify, each vertex uniquely from the ordering of the three surfaces around it. Observe the clockwise ordering of surfaces, denoted by red, green, blue ordering of corners around each vertex in Fig. 5.7.

5. Each intersection curve may contribute zero, one, or more edges. We store each intersection curve uniquely and represent an edge implicitly using parametric values of that edge's start and end point on the corresponding curve.

We thus propose the following data structure to store the three representations of the hub:

- A table $\mathbf{Q}[\ ]$, wherein each entry corresponds to a quador surface. Each quador surface $\mathbf{Q}[i]$ stores:

  - a, b, c: coefficients of a *symmetric conic* profile given by $ax^2 + y^2 + 2bx + c = 0$. It follows from the construction of a quador-beam in a symmetric frame, as given in [1].
  - $\mathbf{F}(O, \overrightarrow{I}, \overrightarrow{J}, \overrightarrow{K}, s)$: a similarity frame, which we define as a frame of three orthonormal vectors $\overrightarrow{I}, \overrightarrow{J}, \overrightarrow{K}$, origin O and a scaling factor s. For the similarity frame associated with a quador surface $\mathbf{Q}_i$, its origin O is at the middle of the centers of the two balls defining $\mathbf{Q}[i]$, vector $\overrightarrow{I}$ is oriented from hub ball's center to the center of the other ball defining $\mathbf{Q}[i]$.

- to improve the uniformity of representations across the data structure, the surface of the ball of the hub is stored as the quador surface $\mathbf{Q}[0]$ with coefficients $a = 1, b = 0, c = -r^2$, where r is the radius of the ball.In our implementation, we have made this deliberate choice in favour of uniformity, but in favour of conciseness of the data structure, one can explicitly store the ball of the hub as a sphere rather than a quador surface represented by a profile and a similarity frame.

- A table $\mathbf{Y}[\,]$, wherein each entry $\mathbf{Y}[v]$ corresponds to a vertex v. Each vertex v stores:
  - G: location (x, y, z) coordinates of vertex v
  - $q[3]$: a triplet of indices of the three incident surfaces
  - $t[3]$: a triplet of parameter values denoting vertex position on the three parameterized intersection curves between the three pairs of these three incident surfaces (Fig. 5.8).

- A table $\mathbf{C}[\,][\,]$, wherein each entry $\mathbf{C}[i][j]$ represents the intersection curve between the quador surfaces $\mathbf{Q}[i]$ and $\mathbf{Q}[j]$. We store each intersection curve $\mathbf{C}[i][j]$ (Fig. 5.8) as the three coefficients a, b, c of a symmetric conic and an associated frame $\mathbf{F}(O, \overrightarrow{I}, \overrightarrow{J}, \overrightarrow{K})$.

- A table $\mathbf{N}[\,]$, wherein each entry $\mathbf{N}[c]$ is the index of the next corner in the face, after corner c (Fig. 5.7).

- A table $\mathbf{L}[\,][\,]$, wherein each entry $\mathbf{L}[i][j]$ defines a plane stored as coefficients a, b, c, d of the *general form*, $ax + by + cz + d = 0$ of a plane. Here a, b, c are the direction cosines of the unit normal of the plane and d is the distance of the plane from the origin. For a single hub, we identify the type of plane, from the position of the plane in the L[ ][ ] table, i.e., $\mathbf{L}[0][i], i > 0$ are contact-planes, $\mathbf{L}[i][i], i > 0$ are split-planes, and $\mathbf{L}[i][j], i \neq j$ are intersection-planes. This labeling is relative to the

hub, hence for a lattice with several hubs, one may convert the global id of the plane to a local id with respect to the current hub.

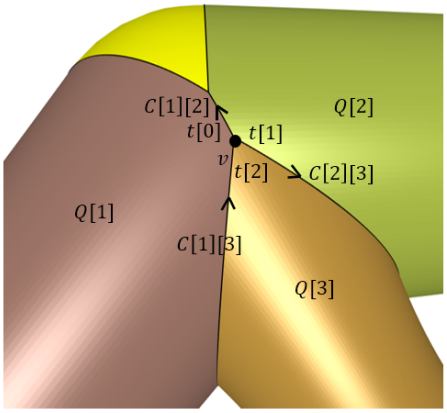


Figure 5.8: An example illustrating that a vertex v has three parameters, $t[0], t[1]$ and $t[2]$, one on each of the three oriented intersection curves between three quador surfaces intersecting at that vertex.

5.4 Constructing the three representations of a hub

We are given a set $\{B_i\}$ of n balls. Where B_0 is the hub's ball, and each pair $(B_0, B_i), i \in [1, n-1]$ forms a quador beam. We compute the three representations of the hub formed at ball B_0, as follows:

5.4.1 Constructing the CSG representation of hub

We construct the CSG of the hub as follows:

- Compute the array $\mathbf{Q}[\,]$ of n quador surfaces corresponding to the hub's ball and the $n-1$ half-beams. Quador surface $\mathbf{Q}[i]$ is oriented such that its axis is in its negative half-space (denoted by $\mathbf{Q}[i]^-$).

- Compute the set of $n-1$ oriented *contact-planes* $\mathbf{L}[0][j],\ j \in (1, n-1)$, of the $n-1$ half beams with the hub's ball. $\mathbf{L}[0][j]$ is oriented such that ball $B[j]$ lies in the

plane's positive half-space (denoted by $\mathbf{L}[0][j]^+$). Also, compute the corresponding plane $\mathbf{L}[j][0]$ by copying from plane $\mathbf{L}[0][j]$, but orient it opposite to $\mathbf{L}[0][j]$.

- Compute the set of $n-1$ oriented *split-planes* $\mathbf{L}[i][i]$, one for each of the half-beams of the hub. Plane $\mathbf{L}[i][i]$ splits the beam corresponding to surface $\mathbf{Q}_i$ into two half-beams and is oriented such that the hub ball's center lies in its positive half-space. For hub's ball, the split plane $\mathbf{L}[0][0] = null$.

A quador half-beam corresponding to surface $\mathbf{Q}_i$ is then represented as $\mathbf{H}_i = \mathbf{Q}_i^- \cap \mathbf{L}[0][i]^+ \cap \mathbf{L}[i][i]^+$ and the hub being the union of the ball and the half-beams is represented by $\mathbf{Q}_0^- \cup \bigcup_{i=1}^{n-1} \mathbf{H}_i$.

5.4.2 Constructing the CST representation of a hub

We assume that we have already computed the oriented contact-planes and the oriented split-planes. We now construct the CST of the hub as follows:

- Compute the set of *intersection-planes* between surfaces of half-beams. Plane $\mathbf{L}[i][j]$, $i, j \neq 0$ and $i \neq j$ is the plane of intersection of quador surfaces $\mathbf{Q}[i]$ and $\mathbf{Q}[j]$. $\mathbf{L}[i][j]$ is oriented such that the ball B_j lies in the plane's positive half-space.

Each quador surface $\mathbf{Q}_i$ is trimmed by a convex polyhedron, given by $\mathbf{T}_i = \bigcap_{j=1}^{n-1} \mathbf{L}[j][i]^+$ and the hub is represented by $closure(\bigcup_{i=0}^{n-1}(\mathbf{Q}_i^- \cap \mathbf{T}_i))$.

5.4.3 Constructing the Brep of a hub

Note that from the CST of the hub, we already have the array $\mathbf{Q}[\,]$ of n quador surfaces and the array $\mathbf{L}[\,][\,]$ of $n \times n$ oriented planes. We now describe, how we compute the vertices and the adjacency graph, i.e. the connectivity information of the Brep of the hub.

In summary, we first compute all the vertices of the hub, and then capture the neighborhood relationships using the data structure described in Sec.5.3. To provide compact

storage, we do not explicitly store edges and faces of the hub, but instead provide operators to efficiently retrieve information required to operate on them. The steps to construct the Brep of the hub are:

- **Compute the 2D array $\mathbf{C}[\,][\,]$ of $n \times n$ conic intersection curves:** Each conic $\mathbf{C}[i][j]$ is the intersection of plane $\mathbf{L}[i][j]$ with surface $\mathbf{Q}[i]$. To avoid multiple representations for the intersection curve, we compute $\mathbf{C}[i][j]$ for $i < j$ and set $\mathbf{C}[j][i] = \mathbf{C}[i][j]$.

- **Compute the table $\mathbf{Y}[\,]$ of vertices:**

  - For each triplet (i, j, k) such that $i < j < k$, compute the line of intersection of the three planes $\mathbf{L}[i][j]$, $\mathbf{L}[j][k]$ and $\mathbf{L}[k][i]$.

  - As each half-beam is trimmed by a convex polyhedron, the angle between normals of planes $\mathbf{L}[i][j]$ and $\mathbf{L}[j][k]$ (or any other pair in cyclic order) is in $[0, \pi]$, hence the line of intersection is oriented consistently along the cross-product of the normals of the planes $\mathbf{L}[i][j]$ and $\mathbf{L}[j][k]$. Refer Fig. 5.9 for an explanation of this claim.

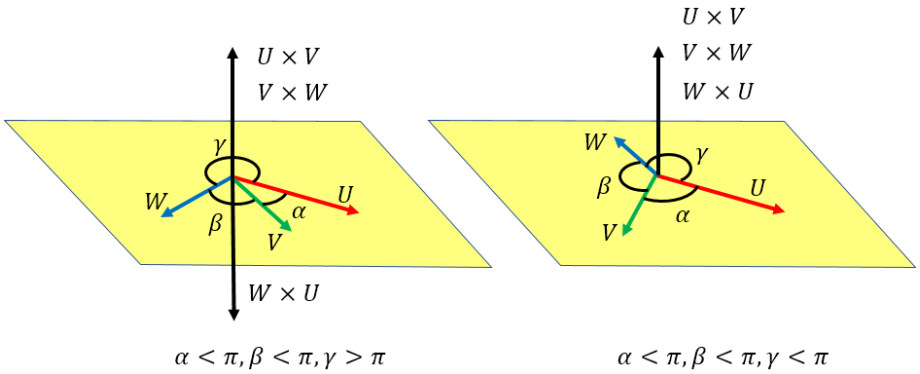


Figure 5.9: If the angle between any cyclic pair of vectors in a set of three coplanar vectors is more than π, then the direction of cross-product from that pair is opposite to that from the other two pairs (left), else the cross-product from each pair is oriented in the same direction

  - This line may intersect neither of the three surfaces, $\mathbf{Q}[i]$, $\mathbf{Q}[j]$ or $\mathbf{Q}[k]$. But if it does, then compute its two intersection points with $\mathbf{Q}[i]$. As the line's orien-

tation is implicit from planes oriented based on surfaces indices, the clockwise order of surfaces (with respect to outward direction at each vertex along the line), at the second intersection point along the line is (i, j, k) and at the first intersection point is (i, k, j) (Fig. 5.6).

- For each of the two intersection points, compute its parametric position on the curves $\mathbf{C}[i][j]$, $\mathbf{C}[j][k]$ and $\mathbf{C}[k][i]$.
- Add two vertices in the $\mathbf{Y}[\,]$ table, one for each of the two points computed above.

- **Compute *loop-edges*, i.e., loops with no vertex**: Some of the intersection curves in the $\mathbf{C}[\,][\,]$ table might not intersect with other curves and will thus form loop-edges (Sec. 5.2.3), i.e. loops with no vertices, e.g. the hub in Fig.5.10 has four loop-edges. As we use a Corner Table based data structure, wherein we traverse through the Brep using corner operators (e.g. next corner on the loop), a loop-edge i.e. a loop without a vertex, won't contribute any corner. Hence, we associate a ***virtual-vertex*** with a loop-edge and add that virtual-vertex to the $\mathbf{Y}[\,]$ table. We store a virtual-vertex with index triplet having repeating indices (i, j, i). A virtual-vertex has no location or parameter triplet. Note that we assume the hub to be of a ***clean-lattice*** (hubs are pairwise disjoint) [3], hence the split-plane always gives a loop-edge, which we represent as a virtual-vertex with index triplet (i, i, i).

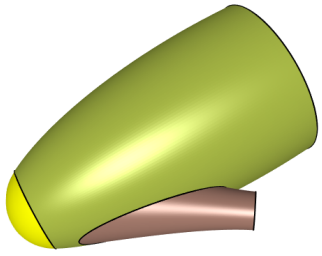


Figure 5.10: Hub with four loop-edges, the ellipse separating the green and brown beams, the circle separating the yellow sphere from the green beam, and the two curves, one each corresponding to the split plane of a half-beam.

- **Prune the $\mathbf{Y}$ table:** Some of the vertices in the $\mathbf{Y}[\,]$ table, might not be on the boundary of the hub (Fig. 5.11). We remove such vertices or virtual-vertices (loops with no vertex) from the $\mathbf{Y}[\,]$ table as follows:

  – A vertex is not on the boundary of the hub if it is contained inside one of the half-beams. As a half-beam is bounded by a surface of revolution, checking vertex containment is reduced to a point-in-circle test in the plane passing through the point and normal to the half-beam's axis.

  – A virtual-vertex (loop without a vertex) is not on the boundary of the hub, if it is inside one of the half-beams. Instead of checking if a loop is contained inside a half-beam, we do a simpler check to see if the loop is outside the trimming polyhedron of one of the half-beams that generated the loop. Which further reduces to checking if a conic lies completely in the negative half-space of a set of oriented planes.

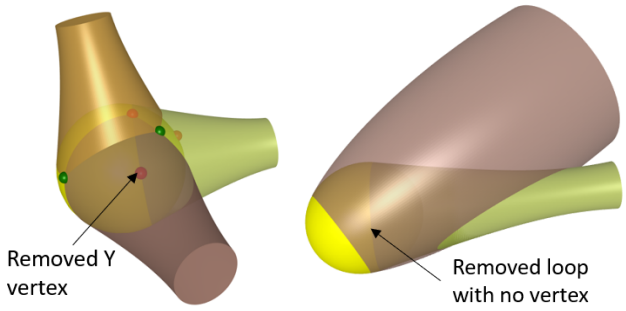


Figure 5.11: Pruning of the Y vertices and of loops with no vertex (loop-edges).

- **Implement some of the corner operators:** We implement certain corner operators, that facilitate the next task of computing the table $\mathbf{N}[\,]$ of next corners.

  - Since each vertex contributes three corners, we compute the vertex of a corner c as $v(c) := \lfloor c/3 \rfloor$. Although a virtual-vertex has only two real corners, for counting we assume that it has three corners. Hence the vertex operator will trivially work for both regular and virtual-vertices.

- For swing operator $s(c)$, we first identify the type of vertex $v(c)$, i.e. a regular Y vertex or a virtual-vertex. We use the repetition of indices in the surface index triplet associated with the vertex to identify the vertex type. Then for a regular vertex, the clockwise swinged corner is $s(c) := 3\lfloor c/3 \rfloor + (c+1)\%3$. For a virtual-vertex, if the associated index triplet is of type (i, j, i), the swing is given by $s(c) := 3\lfloor c/3 \rfloor + (c - 3\lfloor c/3 \rfloor + 1)\%2$, and if the index triplet is (i, i, i), $s(c) := c$. And so, the unswinged corner for a regular vertex is $u(c) := s(s(c))$, and for a virtual-vertex is $u(c) := s(c)$.

- Additionally, we implement operator, $q(c) := \mathbf{Y}[v(c)].q[c\%3]$, which returns the ID of the surface of the corner c, and the operator $t(c) := \mathbf{Y}[v(c)].t[c\%3]$, which returns the parameter of the vertex $v(c)$ on the curve of the edge swung over by $s(c)$,

- **Compute the next corner $\mathbf{N}[c]$:** To compute $\mathbf{N}[c] = k$, for each corner c (Fig.5.12), we search for index $k \in [0, 3 \times \#vertices]$ such that:

  – $q(c) = q(k)$ AND $q(u(c)) = q(s(k))$ and

  – parametric difference $\delta(t(u(c)), t(k))$ is minimum among all k satisfying the above conditions, where the function $\delta(t_{start}, t_{end})$ computes the parametric difference between the corresponding vertices along an open or closed oriented intersection curve.

  – as the curve $\mathbf{C}[i][j], i < j$ is oriented to have surface $\mathbf{Q}[j]$ on its right, when processing corner c to establish $N(c)$, we ensure that $v(c)$ is a start vertex on edge e by requiring that $q(c) > q(u(c))$.

We described above some of the operators that we developed to facilitate the construction of the Brep. In the next section, we describe several other operators to query various elements of a hub's representations.

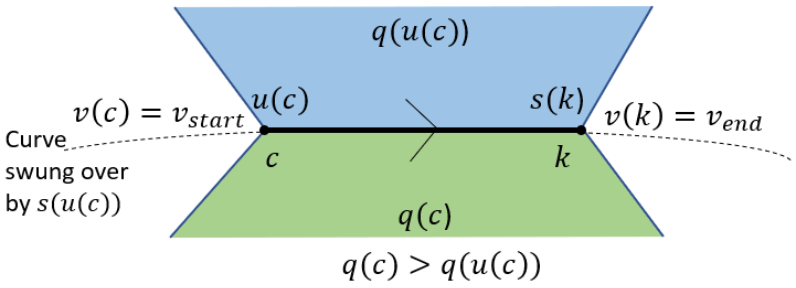


Figure 5.12: Identifying the next corner along the oriented loop.

5.5 Query operators on the three representations of a hub

There are several different queries that can be made on the three representations of a hub, e.g. the loops bounding a face, the edges in a loop, the faces incident on a vertex etc. We propose the following set of query operators.

5.5.1 CSG operators

The following operator queries a CSG primitive. For, $i = 0$, the operator returns the surface of the ball of the hub, else it returns the quador surface, and the contact- and split-plane of the i^{th} half-beam.

- CSG primitive, $h(i) := \mathbf{Q}_i, \mathbf{L}[0][i], \mathbf{L}[i][i]$

5.5.2 CST operators

The following operator queries the CTP of the surface $\mathbf{Q}_i$ of the hub.

- trimming polyhedron of $\mathbf{Q}_i$, $T(i) := \mathbf{L}[j][i] \forall j \in (0, n)$

5.5.3 Brep corner operators

Given a corner c, the following operators are useful to query or traverse the vertices, edges and faces of the Brep of a hub. Refer to Fig. 5.13 for examples of how some of these operators work.

- index of the vertex of the corner, $v(c) := \lfloor c/3 \rfloor$
- index of the quador surface of the corner, $q(c) := \mathbf{Y}[v(c)].q[c\%3]$
- next corner on the loop, $n(c) := \mathbf{N}[c]$
- previous corner on the loop, $p(c) := s(n(s(c)))$
- swing corner around vertex:

  if($q(3v(c)) == q(3v(c)+2)$), $s(c) := 3\lfloor c/3 \rfloor + (c+1)\%2$

  else-if($q(3v(c)) == q(3v(c)+1)$), $s(c) := c$

  else $s(c) := 3\lfloor c/3 \rfloor + (c+1)\%3$
- unswing corner around vertex:

  if($q(3v(c)) == q(3v(c)+2)$), $u(c) := s(c)$

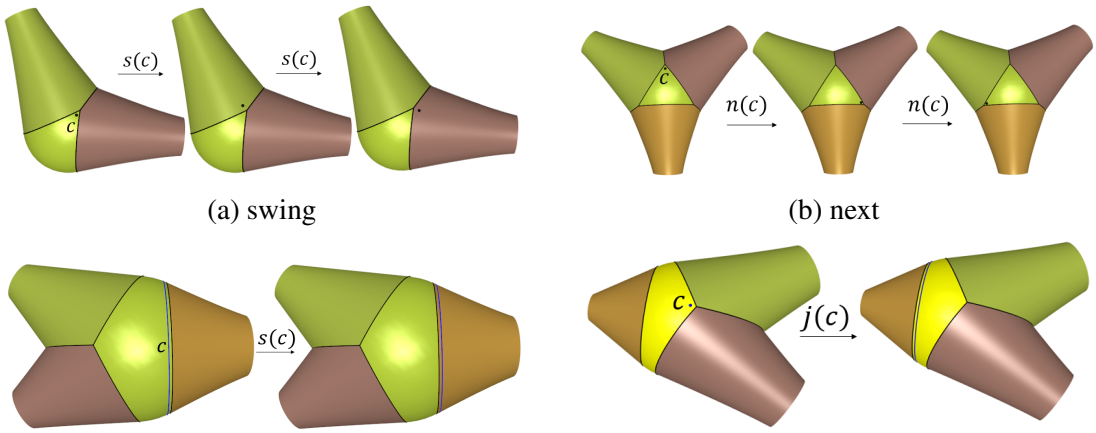
  else $u(c) := s(s(c))$
- parameter of vertex $v(c)$ on the curve of the edge swung over by $s(c)$, $t(c) := \mathbf{Y}[v(c)].t[c\%3]$
- jump to a corner on another loop bounding the same face, $j(c)$: Compute the list of loops bounding $q(c)$. Each loop is a list of corners. Find the loop that contains corner c, and return the first corner of the the next loop in the list of loops.

5.5.4 Brep vertex operators

- one corner of the vertex, $c(v) := 3v$
- location of the vertex, $g(v) := \mathbf{Y}[v].G$

5.5.5 Brep edge operators

We do not store edges as explicit objects in our proposed data structure. Edge between corner c and $n(c)$ can be computed using operator $e(c)$ which returns intersection curve between surfaces $q(u(c))$ and $q(c)$, i.e. parametric conic $\mathbf{C}[q(u(c))][q(c)]$ and parameter values (t_{start}, t_{end}). As the curve is oriented, if $q(c) > q(n(c))$ the edge starts at $v(c)$, else the edge ends at $v(c)$.



(a) swing

(b) next

(c) swing around virtual-vertex (representing a loop-edge). As there is no coordinate location associated with a virtual-vertex, the corner is not displayed by a dot, but by a curve (blue) parallel to the loop-edge, in the face of the corner.

(d) jump from a corner on one loop to a corner on another loop bounding the same face.

Figure 5.13: Examples of corner-based operators for accessing the loops, edges, and vertices of the hub

5.5.6 Brep face operators

For simplicity and efficiency, we define a face $\mathbf{F}$ of a hub $\mathbf{H}$ to be the relative interior of a surface $\mathbf{Q}$ (sphere or beam) that lies on the boundary of the hub. Hence, the face $\mathbf{F}$ of a corner c, may be queried by using the operator $q(c)$ which returns the ID of the surface of c. The face may be conveniently defined either as the intersection of $\mathbf{Q}$ with the Active Zone [46] of the primitive that $\mathbf{Q}$ bounds in the CSG formulation of $\mathbf{H}$ or as the intersection of $\mathbf{Q}$ with the corresponding CTP (Convex Trimming Polyhedron) in the CST formulation. We believe that, for most of the queries on $\mathbf{H}$, these implicit definitions lead to algorithms that are more efficient and more accurate than their counterparts that operate on a boundary representation. Furthermore, we argue that most queries do not require knowing whether the face is connected or, if not (Fig. 5.14), knowing the number of components and their genus. However, we understand that the above knowledge is important for some CAD systems or applications. Hence, we outline here a process for computing that information and point out how it may benefit from the proposed representations of hubs.

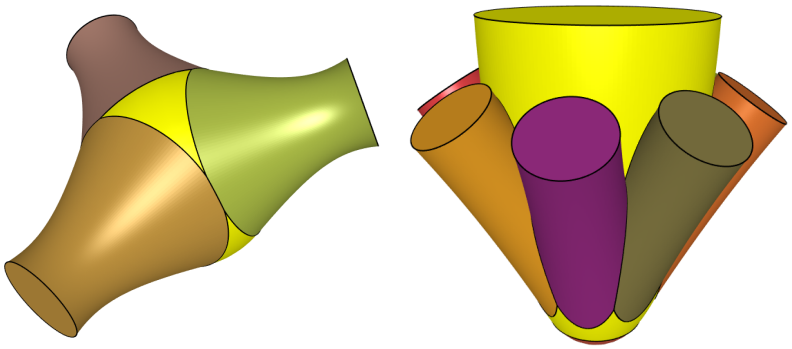


Figure 5.14: Example cases where a face has more than one connected component, the two yellow regions on a ball (left) and the two yellow regions on a beam (right).

We wish to obtain a list of face-components, where, for each component, we have a list of loops, where, for each loop, we store the reference to one corner on that loop. Using the corner-operator next, we can traverse the loop.

There are a variety of approaches for computing the desired output. The approach discussed below may be viewed as the variation of the ***loop-containment-tree*** (see [67] or [68] for computing the shell-containment tree of a connected volume component).

Given the surface $\mathbf{Q}$ that contains face $\mathbf{F}$, we identify a ***base***. When $\mathbf{Q}$ is a half-beam, that *base* (Fig. 5.15) is the circle that bounds the disk where the half-beam is split from its other half. When $\mathbf{Q}$ is the sphere of the hub, the base is a point, which we classify (using the active zone or the CTP) as being inside or outside of $\mathbf{F}$.

For efficiency, we associate each corner c of $\mathbf{F}$ with the ID $\mathbf{L}[c]$ of the loop that passes through the vertex of c. This is precomputed by iterating until all corners of $\mathbf{F}$ are labelled. At each iteration, we increment the loop counter l of $\mathbf{F}$, we pick an unlabeled corner, and we traverse its loop while assigning $\mathbf{L}[c] = l$ for all corners c in that loop. For each loop l of $\mathbf{F}$, we want to compute the label $\mathbf{C}[l]$ of the connected face-component that l bounds. To do so, we will first compute, for each loop l, the ID $\mathbf{P}[l]$ of what we call the parent loop of l. For each non-labelled loop l, we randomly pick a corner c and define an iso-parametric edge $\mathbf{E}$ on $\mathbf{Q}$ that joins vertex $v(c)$ to the base. When $\mathbf{Q}$ is a beam, $\mathbf{E}$ is a line or conic section in a plane that contains $v(c)$ and the axis of the beam. When $\mathbf{Q}$ is a sphere, $\mathbf{E}$

is a great arc on $\mathbf{Q}$. Then, for each other loop, l' of $\mathbf{F}$, we compute the number of times $\mathbf{E}$ crosses l'. (In doing so, we need to distinguish "crossing" from merely "touching".) Using the CST representation, this computation amounts to finding the parameters along $\mathbf{E}$ for points where $\mathbf{E}$ intersects planes that each contain an edge of another loop and sorting these parameters. Let the crossing-count for l' denote the number of times $\mathbf{E}$ intersects l'. If all crossing-counts are even, then we set $\mathbf{P}[l] = 0$. (Note that this identifies the set of loops for face-component $C = 0$.)

More, generally, we associate with loop l the ID $\mathbf{P}[l]$ of the parent loop, which we define as the first loop with odd crossing-count that we cross first while walking from l towards the base along $\mathbf{E}$.

The $\mathbf{P}[\,]$ table defines a loop-containment tree. Its root has depth zero. Each other node (loop) has depth equal to 1 plus the depth of its parent. We compute the depth of each loop. First, we assign a different face-component ID to each node with even depth. Then, to each odd-depth loop, we assign the face-component ID of its parent.

The above process requires a small tweak when $\mathbf{Q}$ is a sphere and the base is a point. If the base is inside $\mathbf{F}$, we proceed as above, but end-up with face-component ID zero not being assigned to any loop. If the base is outside of $\mathbf{F}$, we flip the odd and even adjectives in the above description.

5.6 Experimental validation of our representations for different hubs

We proposed three representations, the CSG, the CST and the Brep of a hub. The CSG representation (union of ball and half-beams, each half-beam being intersection of a quador half-space and two linear half-spaces corresponding to the contact-plane and the split-plane of that beam) is trivial to understand. The CST is built from CSG by using the geometric extents (quador surfaces, contact-planes and split-planes) of the CSG and the intersection-planes between the pairs of half-beams. The Brep is built using the geometric extents of the

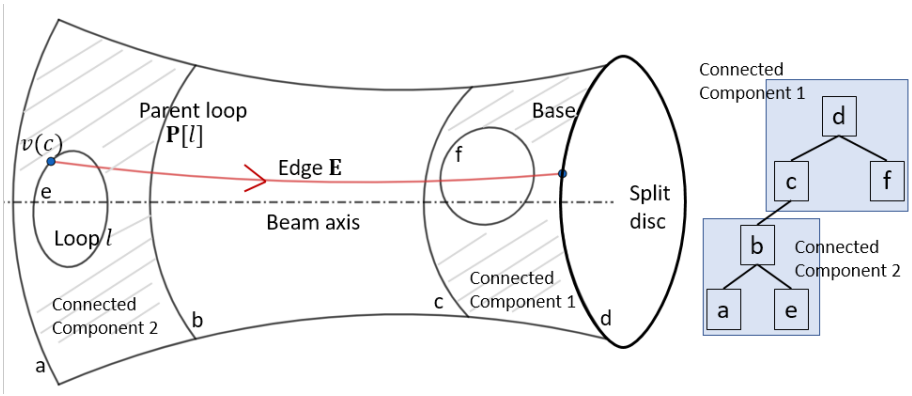


Figure 5.15: Example depicting the process of building the loop-containment tree for a face on a half-beam. It shows loop b, identified as the parent loop of the current loop $l = e$. On the right is the complete loop-containment-tree of all loops, through which we obtain connected components 1 and 2.

CST. As stated earlier in the prior art section of this chapter, several books [44, 41, 45, 42] discuss computing Brep from CSG. We follow similar procedures to compute the Brep of a hub from its CSG (or CST). The three representations of the hub are therefore equivalent, barring the differences due to the numerical errors in computing them.

We have tested our approach (Sec. 5.4) to construct the CSG, CST and Brep of a hub for a variety of hubs. Fig. 5.16 shows some of these hubs. These include simplest of hubs with only one half-beam, to complex hubs with multiple beams, and even cases with multiple beams protruding out of another beam. For each hub, we used the corner operators Sec. 5.5 to correctly traverse the Brep of that hub.

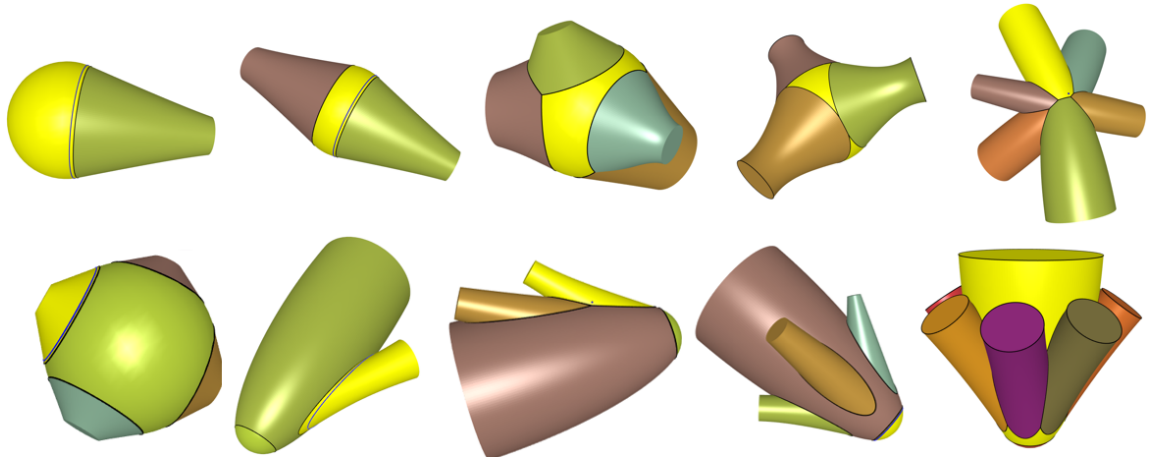


Figure 5.16: A variety of hubs.

In this chapter, we described and provided methods to compute three representations of a hub, the CSG, CST and the Brep of a hub. Although these representations have exact analytical curves and surfaces, their implementations may suffer from topological inaccuracies due to numerical round-off errors, e.g. wrongly classifying a candidate intersection point of three surfaces of the hub as a Brep vertex, when it is not, as it lies inside the another beam of the hub. In the next chapter, we provide a simple and numerically robust algorithm to compute the correct topology of a hub.

CHAPTER 6

A NUMERICALLY ROBUST APPROACH TO COMPUTE THE CORRECT TOPOLOGY OF THE BREP OF A LATTICE WITH QUADOR BEAMS

In this chapter, we propose a **reliable** approach for computing a digital representation of the boundary of a ***lattice*** with quador beams. By reliable, we mean that the topology of the boundary is guaranteed to be ***correct***, even though we are not able to represent exactly the irrational coordinates of its vertices nor the coefficients of parametric models of its edges. These vertices and edges are represented implicitly and their geometry is approximated using floating-point arithmetic.

One of the benefits of decomposing the lattice into hubs (Ch. 4) is that the boundary representation of the overall lattice can be built from that of its hubs, which are interior-disjoint and only touch at disk-faces contained in the split-planes. Building the boundary representation (Brep) of a hub is discussed in Chapter 5. Here, we propose a novel approach for accessing the elements (faces, edges, vertices) of the boundary of a hub (Fig. 6.1) and for traversing the BRep (from one element to an adjacent one) by pivoting in a consistent manner around a vertex or around the boundary of a face.

Most boundary-evaluation algorithms compute the connectivity of the boundary (incidence between faces, edges, and vertices and order of edges around faces and of faces around vertices) by computing numeric approximations of candidate-vertices and by rejecting candidate-vertices that lie inside the sphere or inside any of the quadors. The boundary of a hub comprises its disk faces (where the beams split) and faces that are each contained in the sphere $\mathbf{S_0}$ or in the quador surfaces $\mathbf{Q_i}$ (Fig. 6.1). Hence, the candidate-vertices are each the intersection of three quador surfaces (one or none of which may be $\mathbf{S_0}$). Three quador surfaces in general position may have up to 8 common points, i.e., candidate vertices. Their coordinates are the roots of a system of three quadratic equations. Hence,

known approaches for identifying each candidate vertex and for classifying it (i.e. testing whether it is a vertex of the hub) are either unreliable (because they evaluate the signs of analytical expressions using values that have been corrupted by numeric round-off errors) or computationally expensive and challenging to implement (because they use symbol manipulation or root-separation tools to guarantee that the result of each test is correct). For simplicity and elegance of presentation, we assume general configuration (i.e., that no point lies on four of these quador surfaces).

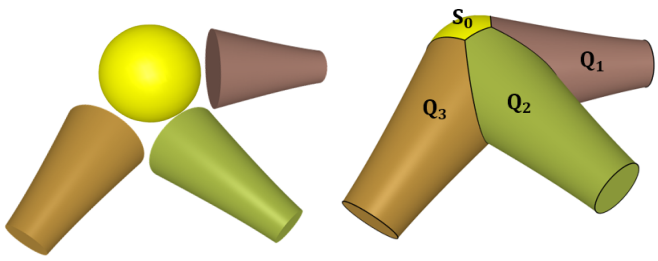


Figure 6.1: A ball and the three half-beams incident on that ball (left). The ball and the half-beams are shown disjoint only to visualize them clearly. In their original position, the half-beams abut the ball and their union forms a hub (right). Each differently colored solid component in the right subfigure is a chunk. $\mathbf{S_0}$ is the surface of the ball, and $\mathbf{Q_1}$, $\mathbf{Q_2}$, and $\mathbf{Q_3}$ are the quador surfaces of the three beams.

We propose a novel boundary-evaluation algorithm that we conjecture is reasonably simple to implement and is reliable (guarantees that the result of each classification is correct). Our approach takes advantage of the following special properties of a hub:

1. The intersection of two quador surfaces of a hub always lies in two planes.
2. The hub is the union of solid components, which we call ***chunks*** (Fig. 6.1), each being the intersection of a quador half-space with a ***convex trimming polyhedron***.

We exploit these special properties of a hub to compute the boundary on each quador surface (and on the surface of the ball of the hub) as an *arrangement of linear half-spaces on the quadric surface*, which is much simpler to compute than the *arrangement of quadrics on a quadric*. Furthermore, our approach requires computing the signs of closed-form algebraic expressions that involve only rational numbers and up to five non-nested square

roots, which we do using only integers and only the plus, minus, multiplication and division operations without ever evaluating a square root.

In our approach, we build a ***trimming complex*** for the hub, wherein each ***room*** of the complex is the interior of the trimming polyhedron of a quador surface of the hub. The rooms are separated by walls and exactly two rooms are incident on each wall. Fig. 6.2 shows a hub and its trimming complex. We discuss building a numerically robust ***dart*** representation [69, 14] of this complex and provide dart operators to traverse the complex. We then build ***corner*** operators [13] using the dart operators of the complex, to traverse the corner based ***boundary representation (Brep)*** of the hub. Thus, in summary, we associate the dart representation of the trimming complex as a proxy representation of the hub and use it to capture the correct topology of the hub. We provide numerically robust tests to build this association between a hub and its trimming complex.

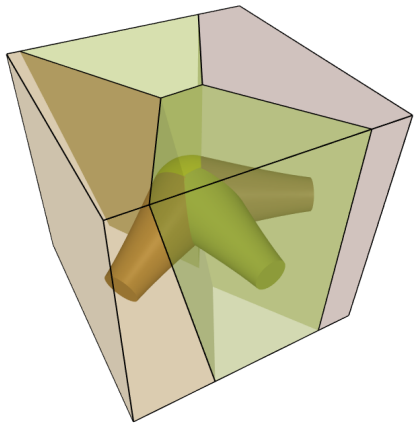


Figure 6.2: A hub and its trimming complex. Each room of the complex is shown in a different color.

The rest of the chapter is organised as follows. In Sec. 6.1, we discuss prior art on computing a numerically robust boundary representations of solids. In Sec. 6.2, we discuss the field of rational numbers and its ***field extensions***, i.e. numbers involving non-nested square roots. We discuss performing simple arithmetic operations on numbers in these field extensions and present a method to evaluate the sign of these numbers without computing the square roots. In Sec. 6.3, we present expressions for the quador surfaces and

planes of a hub and investigate the algebraic nature of the coefficients in these expressions in terms of the number of non-nested square roots they contain. We report the complexity of our approach to compute the correct topology of the Brep of a hub, in terms of the maximum number of non-nested square roots in any expression involved in our computations. In Sec. 6.4, we discuss the complexities associated with the standard approach of generating candidate vertices and testing them to build the Brep. In Sec. 6.5 and Sec. 6.6, we present a novel approach to compute the correct topology of a hub. Our approach is simple and numerically robust. First in Sec. 6.5, we describe the ***trimming complex*** of a hub and discuss numerically robust methods to compute its dart representation. Then in Sec. 6.6, we build the operators to traverse Brep of a hub indirectly by traversing the dart representation of the trimming complex. Our approach relies on three topological tests, In Sec. 6.7, we present numerically robust algorithms to perform these tests using numbers in field extensions. We conclude this chapter by summarising the proposed approach and of its implications in reliable processing of lattices with quador beams.

6.1 Prior art on computing the correct topology of a Brep

Several authors discuss computing the boundary representation (Brep) of a solid model from its CSG representation [44, 70, 41]. This requires performing regularized Boolean operations on CSG primitives to compute the geometry of the Brep elements namely the vertices, edges and faces and use this geometric information to compute the topological relationship between these elements, such as faces incident on an edge, edges bounding a face and edges incident on a vertex. Due to numerical errors in floating-point arithmetic [71] the resulting Brep may have both geometric and topological errors. For example, the coordinates of a vertex at the intersection of three edges may not match, when the vertex is computed as intersection of two different pairs of edges from these three edges. Similarly, when computing the Brep of a convex polyhedron from a set of oriented planes [53], a candidate intersection point of three planes may be wrongly classified against a fourth

plane.

For computing the Brep of polyhedral solids, Sugihara et al. [72] assume that the coefficients of input planes are are rational. They use fixed-precision numbers to represent these coefficients and based on arithmetic operations involved in computing the Brep, guarantee certain numerical precision for the coordinates of the vertices of the Brep. Banarjee et al. [53] use a similar approach to guarantee topological correctness of the Brep of a polyhedral solid. Benouamer et al. [51] combine interval-arithmetic [73] and symbolic computation [74] to make decisions using only the sign of symbolic expressions and delay precise numeric computations till they become either useless or unavoidable.

For computing the Brep of curved solids, Keyser et al. [50, 75] use exact computations [76], wherein instead of computing and storing the numeric values of coordinates of a Brep vertex, the vertex is expressed as intersection of two algebraic curves [77]. They recommend using their solid modeling system ESOLID [50] for low degree solids.

Of the simplest of curved solids are those bounded by quadric surfaces. Dupont et al. [57] propose computing exact intersections between quadric surfaces. They assume that the quadric coefficients are rationals, represent numbers as algebraic expression with square roots, and use only the standard arithmetic operators in their computations.

In this chapter, our goal is to compute the (numerically) correct topology of the Brep of a hub. A hub is a special arrangement of quadric surfaces. One of the surface is a sphere and all others are quadrics-of-revolution (quadors). Each quador surface is tangent to the sphere in a circle, and each pair of quadors intersect in 2 conics. Additionally, we assume general position, i.e. each vertex of the Brep is intersection of three surfaces. In our approach, we compute signs of closed-form algebraic expressions that involve only rational numbers and up to five non-nested square roots. We do so using only standard arithmetic operators.

6.2 Field of rational numbers, and field extensions of numbers with square roots

In our approach (Sec. 6.5 and Sec. 6.6) to compute the correct topology of the Brep of a hub, we evaluate the signs of numbers with non-nested square roots. In this section we describe how we represent such numbers and evaluate their signs.

Let $\mathbb{Q}$ be the field (set) of rational numbers $(a, b, c, d, e, \ldots)$. Then for $\lambda_i \in \mathbb{Q}^+$ (positive rational numbers), $\mathbb{Q}(\sqrt{\lambda_1})$ denotes the field extension of $\mathbb{Q}$, i.e. the set of numbers of the form $a + b\sqrt{\lambda_1}$, where $a, b \in \mathbb{Q}$. Similarly, $\mathbb{Q}(\sqrt{\lambda_1}, \sqrt{\lambda_2})$ is the set of numbers of the form $a + b\sqrt{\lambda_1} + c\sqrt{\lambda_2} + d\sqrt{\lambda_1}\sqrt{\lambda_2}$, $\mathbb{Q}(\sqrt{\lambda_1}, \sqrt{\lambda_2}, \sqrt{\lambda_3})$ is the set of numbers of the form $a + b\sqrt{\lambda_1}+c\sqrt{\lambda_2}+d\sqrt{\lambda_3}+e\sqrt{\lambda_1}\sqrt{\lambda_2}+f\sqrt{\lambda_1}\sqrt{\lambda_3}+g\sqrt{\lambda_2}\sqrt{\lambda_3}+h\sqrt{\lambda_1}\sqrt{\lambda_2}\sqrt{\lambda_3}$ and so on. Note that the numbers of the field $\mathbb{Q}(\sqrt{\lambda_1}, \sqrt{\lambda_2})$ can be written as $(a+b\sqrt{\lambda_1})+(c+d\sqrt{\lambda_1})\sqrt{\lambda_2}$, thus we can write a ***recursive*** definition, where the numbers in $\mathbb{Q}(\sqrt{\lambda_1}, \sqrt{\lambda_2})$ are of the form $a + b\sqrt{\lambda_2}$, where $a, b \in \mathbb{Q}(\sqrt{\lambda_1})$, or in general the numbers in $\mathbb{Q}(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_k})$ is the set $\{a + b\sqrt{\lambda_k}, a, b \in \mathbb{Q}(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_{k-1}})\}$.

6.2.1 Evaluating the signs of numbers in field extensions

With the recursive description of the field extensions, the number $a + b\sqrt{\lambda_k}$ with $a, b \in \mathbb{Q}(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_{k-1}})$ is:

- zero, if $(a = b = 0)$ or if the signs of a and b are opposite and $a^2 = b^2\lambda_k$.
- positive, if $(sign(a) = sign(b) = +)$ or $(sign(a) = +$ and $a^2 > b^2\lambda_k)$, or $(sign(b) = +$ and $a^2 < b^2\lambda_k)$.
- negative, if $(sign(a) = sign(b) = -)$ or $(sign(a) = -$ and $a^2 > b^2\lambda_k)$, or $(sign(b) = -$ and $a^2 < b^2\lambda_k)$.

In practice, to accelerate computations, we recommend using interval arithmetic [78] to compute an evaluation interval for $a + b\sqrt{\lambda_k}$ (wherein we do compute the square root). If the interval does not contain zero then we know the sign of this number and we don't recurse further, else the evaluation interval contains zero, and we recurse to the next level

($\mathbb{Q}(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_{k-1}})$). Then repeat this process on that level and so on. Alternatively, we may avoid interval arithmetic and evaluate the sign of $a + b\sqrt{\lambda_k}$ by recursing all the way to level $k = 0$.

6.2.2 Arithmetic operations on numbers in field extensions

For adding, subtracting, multiplying and dividing two numbers, one number in $\mathbb{Q}(\sqrt{\lambda_i}, \ldots, \sqrt{\lambda_m})$ and the other number in $\mathbb{Q}(\sqrt{\lambda_i}, \ldots, \sqrt{\lambda_n})$, the resulting number is in $\mathbb{Q}(\{\sqrt{\lambda_i}, \ldots, \sqrt{\lambda_m}\} \cup \{\sqrt{\lambda_i}, \ldots, \sqrt{\lambda_n}\})$. For example, let $p = a + b\sqrt{\lambda_1}$ be a number in $\mathbb{Q}(\sqrt{\lambda_1})$ and $q = c + d\sqrt{\lambda_2} + e\sqrt{\lambda_3} + f\sqrt{\lambda_2}\sqrt{\lambda_3}$ be a number in $\mathbb{Q}(\sqrt{\lambda_2}, \sqrt{\lambda_3})$, then

- **Addition/Subtraction:** $p+q = (a+c)+b\sqrt{\lambda_1}+d\sqrt{\lambda_2}+e\sqrt{\lambda_3}+f\sqrt{\lambda_2}\sqrt{\lambda_3}$, therefore, $p + q \in \mathbb{Q}(\sqrt{\lambda_1}, \sqrt{\lambda_2}, \sqrt{\lambda_3})$.
- **Multiplication:** $p \times q = ac + bc\sqrt{\lambda_1} + ad\sqrt{\lambda_2} + ae\sqrt{\lambda_3} + af\sqrt{\lambda_2}\sqrt{\lambda_3}$, thus, $p \times q \in \mathbb{Q}(\sqrt{\lambda_1}, \sqrt{\lambda_2}, \sqrt{\lambda_3})$.
- **Inverse:** $1/p = 1/(a + b\sqrt{\lambda_1}) = (a - b\sqrt{\lambda_1})/(a^2 - b^2\lambda_1)$, thus, $1/p \in \mathbb{Q}(\sqrt{\lambda_1})$.
- **Division:** $q/p = q \times (1/p)$, which is a product of a number in $\mathbb{Q}(\sqrt{\lambda_2}, \sqrt{\lambda_3})$ with a number in $(\mathbb{Q}(\sqrt{\lambda_1}))$, therefore, $p/q \in \mathbb{Q}(\sqrt{\lambda_1}, \sqrt{\lambda_2}, \sqrt{\lambda_3})$.

Clearly, when there is a common factor, for example, $p \in \mathbb{Q}(\sqrt{\lambda_1})$ and $q \in \mathbb{Q}(\sqrt{\lambda_1}, \sqrt{\lambda_2})$ the result of addition/subtraction/multiplication or division of p and q is in $\mathbb{Q}(\sqrt{\lambda_1}, \sqrt{\lambda_1}, \sqrt{\lambda_2})$ which is a number in $\mathbb{Q}(\sqrt{\lambda_1}, \sqrt{\lambda_2})$.

6.3 A hub

In this section, we provide expressions for the ***geometric extents***, namely the quador surfaces and trimming planes of a hub. We discuss the nature of the coefficients in these expressions, in terms of the number of non-nested square roots they contain, i.e. the field extensions they belong to. In subsequent sections, we assess the complexity of our approach to compute the correct topology of the Brep of a hub, in terms of the maximum number of non-nested square roots in any expression involved in our computations.

6.3.1 Geometric extents of a hub

Let $\mathbf{S_0}$ be the sphere of a hub. We define a set of quador surfaces $\{\mathbf{Q_i}, i \in [1, n]\}$, each $\mathbf{Q_i}$ being tangent to the spheres $\mathbf{S_0}$ and $\mathbf{S_i}$ in ***contact-planes***. The shape of each quador $\mathbf{Q_i}$ is controlled by corresponding real valued coefficient λ_i. Two quador surfaces of a hub intersect in two ***intersection-planes***. Each quador surface is split into two halves by a ***split-plane*** (Sec.4.1) . Fig. 6.3 shows the geometric extents of a simple hub:

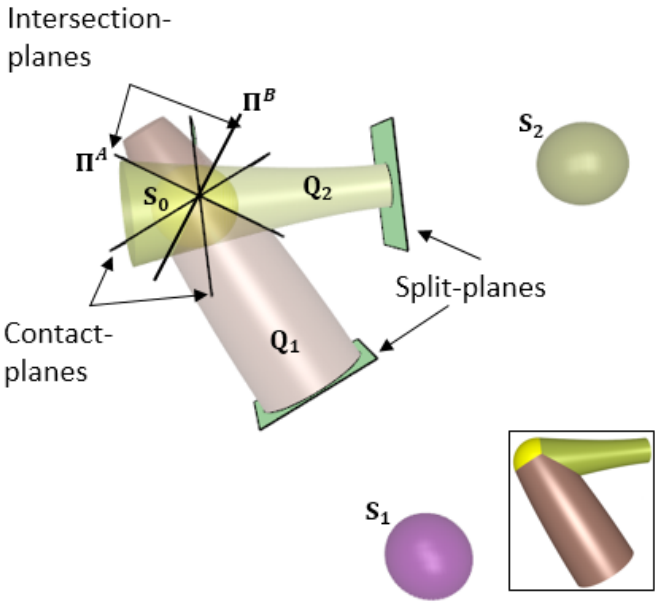


Figure 6.3: Geometric extents of a simple hub. $\mathbf{S_0}$ is the sphere of the hub. $\mathbf{Q_1}$ and $\mathbf{Q_2}$ are each tangent to $\mathbf{S_0}$ in a contact-plane and are each split by a split-plane. $\mathbf{Q_1}$ and $\mathbf{Q_2}$ intersect in two intersection-planes $\mathbf{\Pi^A}$ and $\mathbf{\Pi^B}$. For each $\mathbf{Q_i}$, the the closure of the intersection of its tubular (open) half-space and the linear (open) half-spaces corresponding to its contact-plane and split-plane, defines a half-beam. The hub (thumbnail) is the union of the $\mathbf{S_0}$ and the half-beams incident on it.

6.3.2 Assumptions and simplifications

We make the following assumptions and corresponding simplifications about the hub:

– The coordinates of the centers of the spheres, the radii of the spheres and the scalar coefficients ($\{\lambda_i\}$) controlling the shapes of the beams are all rational numbers.

– We assume that the lattice is clean, i.e., the hubs are pairwise interior-disjoint. Two hubs with a common beam may touch in the disk splitting that beam.
– The split-planes are introduced by us to define a half-beam. They do not affect the topology of a hub. Therefore, for simplicity, we disregard the split-planes in our approach to compute the correct topology of a hub.
– The two oriented intersection-planes (e.g. $\mathbf{\Pi^A}, \mathbf{\Pi^B}$ in Fig. 6.3) of a pair of quador surfaces are defined implicitly from their oriented contact-planes [1]. As each quador is trimmed by its oriented contact-plane, only the intersection-plane $\mathbf{\Pi^A}$ is relevant in our computations [1].
– The half-beams of a hub form a singly connected component, and that no half-beams sticks out of another half-beam. These design constraints are desirable to strengthen the hub as a lattice junction where multiple half-beam meet the ball and to avoid structurally weak junctions such as one isolated half-beam with the ball or one half-beam with another half-beam. With these assumptions, the Brep of a hub with n half-beams have n closed-loop circular edges, one per beam as the intersection of its surface with its split-plane, and the set of all other edges is connected. Fig. 6.4 shows valid, and invalid hubs under these assumptions.

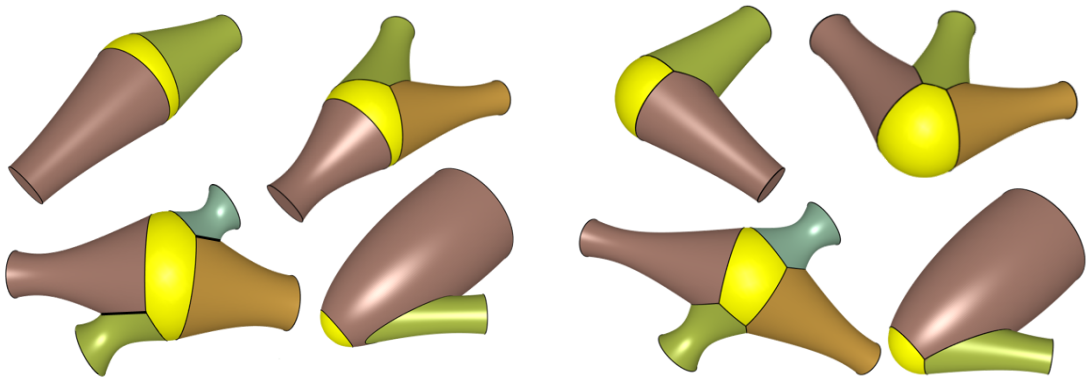


Figure 6.4: In the left subfigure, in the top-left, top-right and bottom-left hubs, the half-beams do not form a connected component, and in the bottom-right hub one beam sticks out of the other. We assume that such configurations are undesirable to avoid structurally weak spots in the hub, hence we exclude such hubs from our scope. In the right subfigure, the four hubs satisfy our assumptions.

– A contact-plane or an intersection-plane trims two surfaces, one on its either side. To consistently differentiate between these two sides, we associate two expressions with each contact- or intersection-plane. These expressions define two oppositely oriented planes Fig. 6.5. For example, for the hub of Fig. 6.3, we have 4 expressions ($\mathbf{\Pi_{01}}$, $\mathbf{\Pi_{10}}$, $\mathbf{\Pi_{02}}$, $\mathbf{\Pi_{20}}$) of oriented contact-planes and 2 expressions ($\mathbf{\Pi_{12}}$,$\mathbf{\Pi_{21}}$) of oriented intersection-planes. Furthermore, we denote the positive half spaces associated with an oriented plane, say e.g. plane $\mathbf{\Pi_{12}}$ as $\mathbf{\Pi^{+}_{12}}$. The order of indices in the subscript denote the orientation of the plane, e.g., $\mathbf{\Pi_{10}}$ is oriented to have $\mathbf{S_0}$ in its positive half-space $\mathbf{\Pi^{+}_{10}}$, while $\mathbf{\Pi_{01}}$ is oriented to have $\mathbf{S_1}$ in its positive half-space $\mathbf{\Pi^{+}_{01}}$.

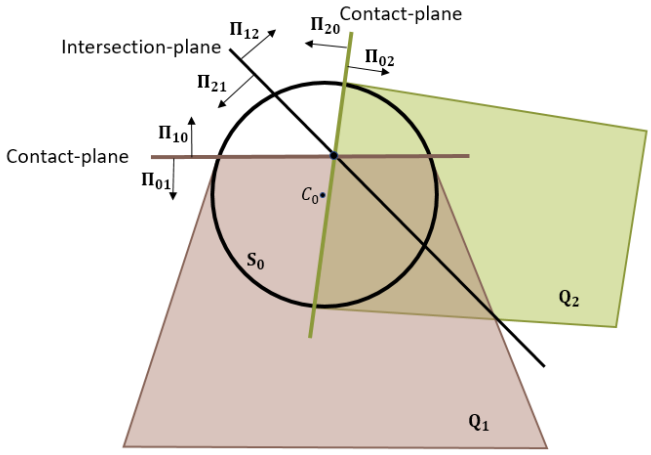


Figure 6.5: A 2D schematic cross-section of a hub. With each contact-plane or intersection-plane we associate two expressions that define two oppositely oriented planes and that the order of indices in the subscript denotes the orientation of the plane.

6.3.3 Expressions for the geometric extents of a hub

In this section, we present the expressions for the geometric extents (surfaces and planes) of a hub. We determine if the coefficients in these expressions are rational numbers or are numbers with square roots:

– Quador surface $\mathbf{Q_i}$ connects tangentially to spheres $\mathbf{S_0}$ (center C_0, radius r_0) and $\mathbf{S_i}$ (center C_i, radius r_i). The expressions of the oriented contact-planes $\mathbf{\Pi_{0i}}$ of $\mathbf{Q_i}$ with

$\mathbf{S_0}$ is:

$$\overrightarrow{C_0P}\cdot\overrightarrow{C_0C_i} - \frac{\overrightarrow{C_0C_i}^2 + r_0^2 - r_1^2}{2} + \frac{\overrightarrow{C_0C_i}^2}{2\lambda_i} \qquad (6.1)$$
$$\text{and, } \mathbf{\Pi_{i0}} \text{ is simply } -\mathbf{\Pi_{0i}}$$

As λ_i is rational and the coordinates of C_0 and C_i are rational (assumptions stated in Sec. 6.3.2), the coefficients in the expressions for oriented contact-planes ($\mathbf{\Pi_{0i}}$ or $\mathbf{\Pi_{i0}}$) $\in \mathbb{Q}$.

– The expression for surface $\mathbf{Q_i}$ (Sec. 4.2.1) is:

$$\mathbf{S_0} + \lambda_i {\mathbf{\Pi_{0i}}}^2 \qquad (6.2)$$

Hence, the coefficients in the expressions for $\mathbf{Q_i} \in \mathbb{Q}$.

– For a pair of quador surfaces $\{\mathbf{Q_i}, \mathbf{Q_j}\}$, the expression for intersection-plane $\mathbf{\Pi_{ji}}$ (Sec. 4.2.1) is:

$$\sqrt{|\lambda_i|}\mathbf{\Pi_{0i}} - \sqrt{|\lambda_j|}\mathbf{\Pi_{0j}} \qquad (6.3)$$
$$\text{and, } \mathbf{\Pi_{ij}} \text{ is simply } -\mathbf{\Pi_{ji}}$$

Hence, the coefficients in the expressions for oriented intersection-planes ($\mathbf{\Pi_{ij}}$ or $\mathbf{\Pi_{ji}}$) $\in \mathbb{Q}(\sqrt{\lambda_i}, \sqrt{\lambda_j})$, i.e. the field extension of numbers with two non-nested square roots.

In the next section, we discuss a standard approach for computing the topology of the Brep of a hub. This approach is numerically less robust as it may involve evaluating expres-

sions with nested square roots, therefore, we present a more robust approach in subsequent sections.

6.4 Standard approach for computing the correct topology of the Brep of a hub

A standard approach for computing a B-Rep of a hub is to compute for each quador surface $\mathbf{Q_i}$ in turn, the boundary of the faces of the hub supported by $\mathbf{Q_i}$. In an iterative approach, i.e. maintaining the B-Rep of $\mathbf{Q_i}$ while adding the half-beams one by one, the standard method requires to test whether a candidate vertex (intersection of $\mathbf{Q_i}$, $\mathbf{Q_j}$, and $\mathbf{Q_k}$) lie in, out or on a fourth quador surface $\mathbf{Q_l}$, that is in, out or on quador volumes. Evaluating these tests exactly is a priori difficult because the vertices are intersection points of three quador surfaces and they are thus the solutions of degree-eight algebraic systems of three equations of degree 2. As we will see later (Sec. 6.5.2), these vertices are actually the intersection points of the quadric $\mathbf{Q_i}$ with a line that can be parameterized with coefficients involving at most three non-nested square roots. Thus the coordinates of these points can be expressed with square roots of numbers in $\mathbb{Q}(\sqrt{\lambda_i}, \sqrt{\lambda_j}, \sqrt{\lambda_k})$. Testing whether these vertices lie in, out or on a fourth quador surface is still difficult because it involves computations with **nested square roots**. In the next two sections, we propose a computationally simpler approach that avoids nested square roots and all computations are done by evaluating the sign of expressions with at most 5 non-nested square roots.

6.5 Trimming Complex

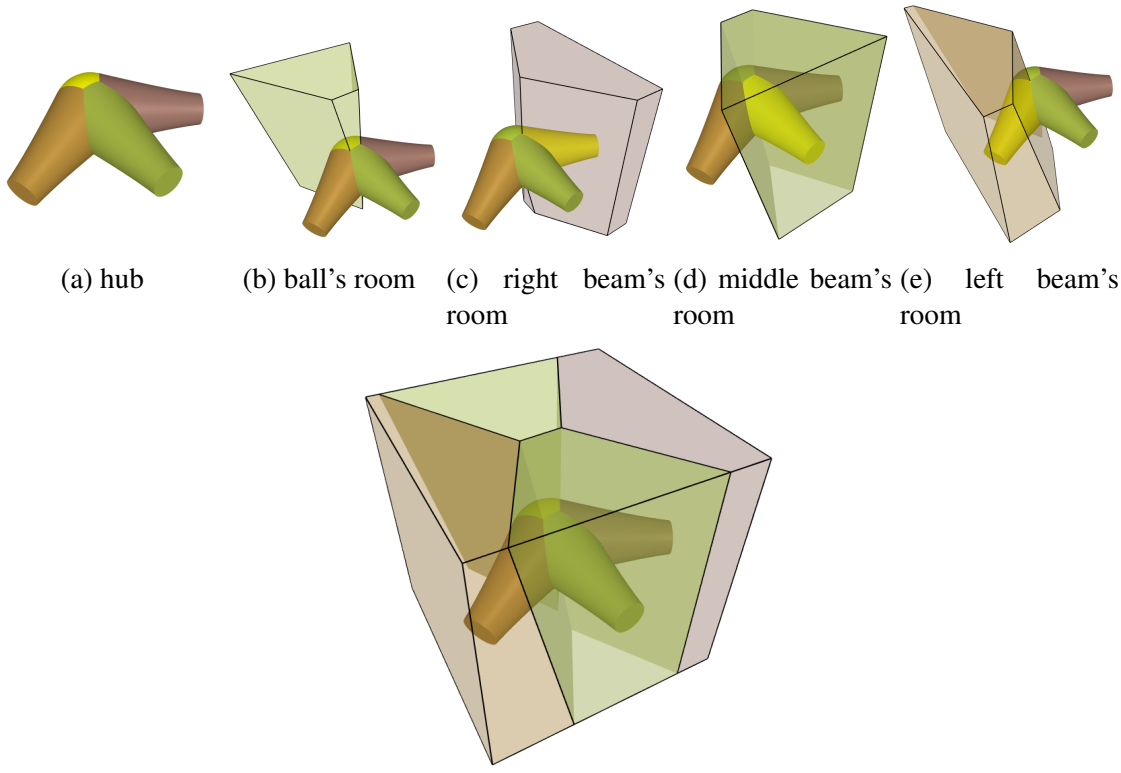
Each quadric surface of a hub is trimmed by a convex polyhedron (Sec. 5.2.2). In this section, we conjecture that the closure of the union of their interiors forms a polyhedral complex, which we call the ***trimming complex*** of the hub (Fig. 6.6). Furthermore, we conjecture that, without split-planes (assumption stated in Sec. 6.3.2), the trimming complex covers the 3D space.

6.5.1 Elements of a Trimming Complex

A trimming complex of a hub consists of the following elements:

- **Room**: open convex trimming polyhedron, one for each quadric surface, and one for the ball of the hub.
- **Wall**: open convex polygon bounding a room.
- **Edge**: oriented open line segment bounding a wall.
- **Vertices**: start and end of an edge.

Fig. 6.6 shows a hub, its trimming complex, and (closure of) individual rooms of the complex. Note that as the complex covers the 3D space, for better visualization, we show the complex trimmed by a box bounding the hub.



(a) hub (b) ball's room (c) right beam's room (d) middle beam's room (e) left beam's room

(f) trimming complex and the hub

Figure 6.6: A hub, (closure of) rooms of its ball and each of its beams, and the trimming complex formed by the closure of the union of these rooms.

In the next section, we prove special properties of the contact-planes and intersection-

planes of a hub. Later in Sec. 6.5.3, we use these properties to conjecture that the closure of the union of the rooms of the ball and the quadors of a hub forms a polyhedral complex and that this complex covers the whole 3D space.

6.5.2 Special properties of the arrangement of contact-planes and of the intersection-planes of a hub

We prove the following theorems about the arrangement of contact-planes and the intersection-planes of a hub.

Theorem 6.5.1. *For two quador surfaces* $\mathbf{Q_i}$ *and* $\mathbf{Q_j}$ *of a hub, where trimmed* $\mathbf{Q_i}$ *and* $\mathbf{Q_j}$ *are contained in (open) half-spaces* $\mathbf{\Pi^+_{0i}}$ *and* $\mathbf{\Pi^+_{0j}}$ *respectively, the intersection curve of the trimmed* $\mathbf{Q_i}$ *and* $\mathbf{Q_j}$ *is contained in the plane* $\mathbf{\Pi_{ij}} = \sqrt{\lambda_j}\mathbf{\Pi_{0j}} - \sqrt{\lambda_i}\mathbf{\Pi_{0i}}$.

Though proven earlier in Sec. 4.2.1, for convenience, we reproduce it as follows:

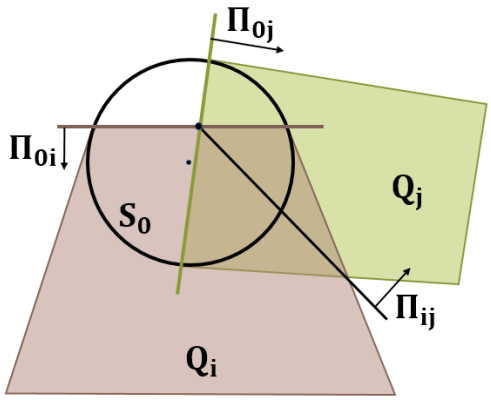


Figure 6.7: A 2D schematic cross-section of a hub. Oriented intersection-plane $\mathbf{\Pi_{ij}}$ is computed implicitly from oriented contact-planes $\mathbf{\Pi_{0i}}$ and $\mathbf{\Pi_{0j}}$.

Proof. Consider the hub of Fig. 6.7. There exist real numbers λ_i and λ_j such that $\mathbf{Q_i} = \mathbf{S_0} + \lambda_i\mathbf{\Pi^2_{0i}}$ and $\mathbf{Q_j} = \mathbf{S_0} + \lambda_j\mathbf{\Pi^2_{0j}}$. Where $\mathbf{Q_i}$ and $\mathbf{Q_i}$ intersect, $\mathbf{Q_i} = \mathbf{Q_j}$, i.e., $\mathbf{S_0} + \lambda_i\mathbf{\Pi^2_{0i}} = \mathbf{S_0} + \lambda_j\mathbf{\Pi^2_{0j}}$, i.e., $\lambda_j\mathbf{\Pi^2_{0j}} - \lambda_i\mathbf{\Pi^2_{0i}} = 0$. Upon factorization we obtain:

$$(\sqrt{|\lambda_j|}\mathbf{\Pi_{0j}} - \sqrt{|\lambda_i|}\mathbf{\Pi_{0i}})(\sqrt{|\lambda_j|}\mathbf{\Pi_{0j}} + \sqrt{|\lambda_i|}\mathbf{\Pi_{0i}}) = 0$$

As trimmed $\mathbf{Q_i}$ and $\mathbf{Q_j}$ are contained in $\mathbf{\Pi^+_{0i}}$ and $\mathbf{\Pi^+_{0j}}$ respectively, their intersection curve must lie in $\mathbf{\Pi^+_{0i}} \cap \mathbf{\Pi^+_{0j}}$, i.e where $\mathbf{\Pi_{0i}} > 0$ and $\mathbf{\Pi_{0j}} > 0$. Additionally $\sqrt{\lambda_i} > 0$ and $\sqrt{\lambda_j} > 0$. With these constraints, the plane that contains the intersection curve must be obtained by $\sqrt{|\lambda_j|}\mathbf{\Pi_{0j}} - \sqrt{|\lambda_i|}\mathbf{\Pi_{0i}} = 0$, as the other expression $\sqrt{|\lambda_j|}\mathbf{\Pi_{0j}} + \sqrt{|\lambda_i|}\mathbf{\Pi_{0i}}$ cannot evaluate to zero. Furthermore, the intersection-plane is implicitly oriented such that $\mathbf{\Pi_{ij}} = \sqrt{|\lambda_j|}\mathbf{\Pi_{0j}} - \sqrt{|\lambda_i|}\mathbf{\Pi_{0i}}$. □

Corollary 6.5.1.1. *For two quador surfaces of a hub, the two contact-planes (one for each surface), and their intersection-plane are concurrent in a line (Fig. 6.8).*

Proof. We proved above that the expression for the oriented intersection-plane $\mathbf{\Pi_{ij}}$ is $\mathbf{\Pi_{ij}} = \sqrt{|\lambda_j|}\mathbf{\Pi_{0j}} - \sqrt{|\lambda_i|}\mathbf{\Pi_{0i}}$. Hence, when $\mathbf{\Pi_{0i}} = 0$ and $\mathbf{\Pi_{0j}} = \mathbf{0}$, i.e. at the line where the two contact-planes intersect, the intersection-plane $\mathbf{\Pi_{ij}} = 0$. Thus, the the two contact-planes and the intersection-plane are concurrent in a line. □

Corollary 6.5.1.2. *For three quador surfaces of a hub, the three intersection-planes (one for each pair of surfaces) are concurrent in a line (Fig. 6.8).*

Proof. Consider three quador surfaces $\mathbf{Q_i}$, $\mathbf{Q_j}$, and $\mathbf{Q_k}$, and consider the three oriented intersection-planes $\mathbf{\Pi_{ij}}$, $\mathbf{\Pi_{jk}}$ and $\mathbf{\Pi_{ki}}$, then at the line of intersection of $\mathbf{\Pi_{ij}}$ and $\mathbf{\Pi_{jk}}$, $\sqrt{|\lambda_j|}\mathbf{\Pi_{0j}} - \sqrt{|\lambda_i|}\mathbf{\Pi_{0i}} = 0$ and $\sqrt{|\lambda_k|}\mathbf{\Pi_{0k}} - \sqrt{|\lambda_j|}\mathbf{\Pi_{0j}} = 0$, adding the two equations, we obtain $\sqrt{|\lambda_k|}\mathbf{\Pi_{0k}} - \sqrt{|\lambda_i|}\mathbf{\Pi_{0i}} = 0$, which is the equation for $\mathbf{\Pi_{ik}}$ (or $-\mathbf{\Pi_{ki}}$). Thus, the three intersection-planes are concurrent in a line. □

Lemma 6.5.2. *For any i, j, k, the three planes $\mathbf{\Pi_{ij}}, \mathbf{\Pi_{jk}}$ and $\mathbf{\Pi_{ki}}$ have a common line, and the three (open) half-spaces $\mathbf{\Pi^+_{ij}}, \mathbf{\Pi^+_{jk}}$ and $\mathbf{\Pi^+_{ki}}$ have an empty intersection.*

Proof. Following corollaries 6.5.1.1 and 6.5.1.2 for any i, j, k planes $\mathbf{\Pi_{ij}}, \mathbf{\Pi_{jk}}$ and $\mathbf{\Pi_{ki}}$ have a common line. Furthermore, when one of the indices is 0, e.g., $(0, i, j)$, following corollary 6.5.1.1, we may write $\mathbf{\Pi_{ij}} = -(\sqrt{|\lambda_i|}\mathbf{\Pi_{0i}} + \sqrt{|\lambda_j|}\mathbf{\Pi_{j0}})$. In fact, by rearranging the terms we may write, $\mathbf{\Pi_{0i}} = -(a\mathbf{\Pi_{ij}} + b\mathbf{\Pi_{j0}})$, or $\mathbf{\Pi_{ij}} = -(c\mathbf{\Pi_{j0}} + d\mathbf{\Pi_{0i}})$ or $\mathbf{\Pi_{j0}} =$

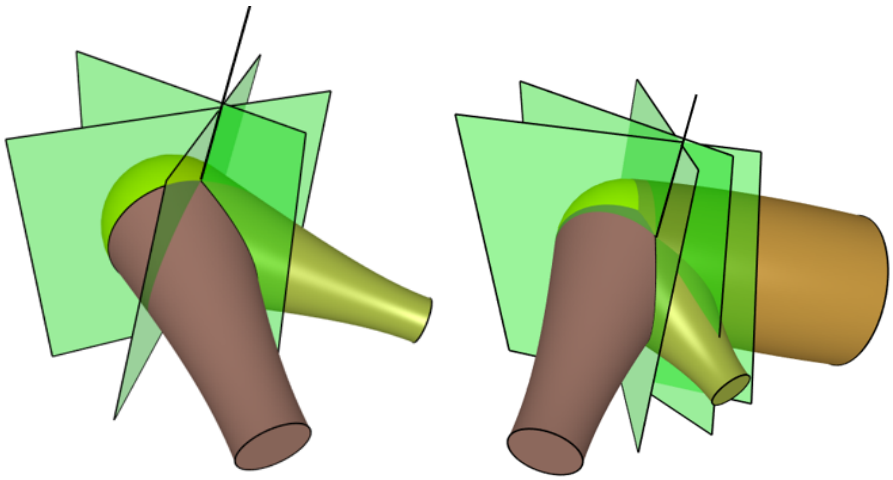


Figure 6.8: A hub with 2 quador beams (left). The contact-planes of the two quador surfaces with the ball and their intersection-plane are concurrent in a line. A hub with 3 beams (right). The three pairwise intersection-planes are concurrent in a line.

$-(e\mathbf{\Pi_{0i}} + f\mathbf{\Pi_{ij}})$, where a, b, c, d, e and f are real positive coefficients. Similarly, for three non zero indices i, j, k, following corollary 6.5.1.2, we may write $\mathbf{\Pi_{ki}} = -(\mathbf{\Pi_{ij}} + \mathbf{\Pi_{jk}})$ or $\mathbf{\Pi_{ij}} = -(\mathbf{\Pi_{jk}} + \mathbf{\Pi_{ki}})$ or $\mathbf{\Pi_{jk}} = -(\mathbf{\Pi_{ki}} + \mathbf{\Pi_{ij}})$. Hence, any one of the three planes, cannot pass through the intersection of the positive half-spaces of the other two, e.g., $\mathbf{\Pi_{ij}}$ cannot not pass through $\mathbf{\Pi^+_{jk}} \cap \mathbf{\Pi^+_{ki}}$, as in there both $\mathbf{\Pi_{ij}} > 0$ and $\mathbf{\Pi_{jk}} > 0$ and thus $\mathbf{\Pi_{ki}} \neq 0$. Similarly, $\mathbf{\Pi_{jk}}$ cannot not pass through $\mathbf{\Pi^+_{ki}} \cap \mathbf{\Pi^+_{ij}}$, and $\mathbf{\Pi_{ki}}$ cannot not pass through $\mathbf{\Pi^+_{ij}} \cap \mathbf{\Pi^+_{jk}}$. Hence, $\mathbf{\Pi^+_{ij}} \cap \mathbf{\Pi^+_{jk}} \cap \mathbf{\Pi^+_{ki}} = \emptyset$. □

Another way of interpreting the above result is that for a plane $ax + by + cz + d = 0$, the direction cosines of its normal $\overrightarrow{N}$ are (a, b, c), thus, $\mathbf{\Pi_{ki}} = -(\mathbf{\Pi_{ij}} + \mathbf{\Pi_{jk}})$ means, their corresponding normals are related by $\overrightarrow{N_{ki}} = -(\overrightarrow{N_{ij}} + \overrightarrow{N_{jk}})$. Thus, each plane is oriented such that the normals of the other two planes lie on the opposite side of its normal, e.g., $\mathbf{\Pi_{ki}}$ is oriented such that the normals $\overrightarrow{N_{ij}}$ and $\overrightarrow{N_{jk}}$ are on opposite sides of $\overrightarrow{N_{ki}}$.

With the above constraints, given planes $\mathbf{\Pi_{ij}}$ and $\mathbf{\Pi_{jk}}$, plane $\mathbf{\Pi_{ki}}$ must lie in the green shaded region and must have orientation as shown in Fig. 6.9.

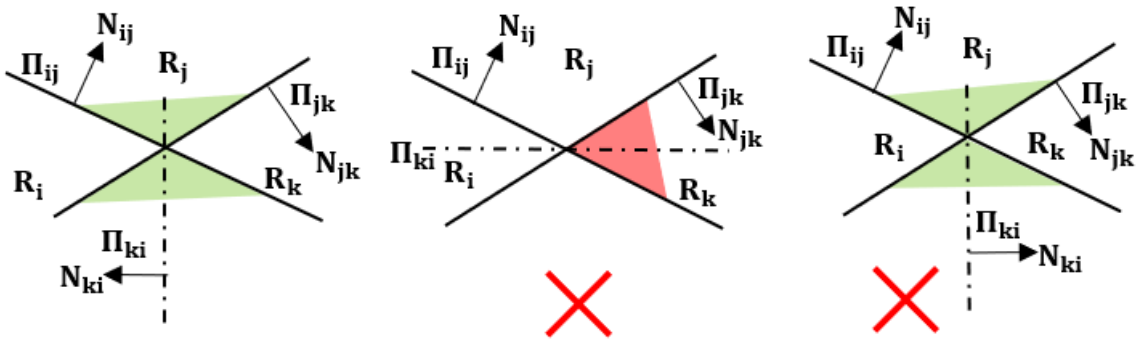


Figure 6.9: For any i, j, k planes $\mathbf{\Pi_{ij}}$, $\mathbf{\Pi_{jk}}$, and $\mathbf{\Pi_{ki}}$ are concurrent in a line. Additionally, as shown in the left subfigure, given $\mathbf{\Pi_{ij}}$ and $\mathbf{\Pi_{jk}}$, $\mathbf{\Pi_{ki}}$ must lie in the green shaded region, and must be oriented such that $\mathbf{\Pi_{ij}^+} \cap \mathbf{\Pi_{jk}^+} \cap \mathbf{\Pi_{ki}^+} = \emptyset$. As shown in the middle subfigure, plane $\mathbf{\Pi_{ki}}$ cannot pass through $\mathbf{\Pi_{ij}^+} \cap \mathbf{\Pi_{jk}^+}$ (the red shaded region), and it cannot be oriented as shown as shown in the right subfigure, as then $\mathbf{\Pi_{ij}}$ passes through $\mathbf{\Pi_{jk}^+} \cap \mathbf{\Pi_{ki}^+}$.

6.5.3 The closure of the union of the rooms of a hub is a polyhedral trimming complex

We conjecture that the closure of the union of the rooms of this hub is a polyhedral trimming complex and that this trimming complex covers the 3D space. The proof of this conjecture is considerably involved and we are still developing it. We discuss below our rationale for our conjectures.

Let $\mathbf{R_0}$ and $\mathbf{R_1}$ are the two rooms of a hub with one half-beam. $\mathbf{R_0}$ is the open half-space $\mathbf{\Pi_{10}^+}$ and $\mathbf{R_1}$ is the open half-space $\mathbf{\Pi_{01}^+}$. The closure of $\mathbf{\Pi_{10}^+} \cup \mathbf{\Pi_{01}^+}$ is a polyhedral complex and covers $\mathbb{R}^3$. Let this be true for rooms $\mathbf{R_0}, ..., \mathbf{R_{n-1}}$ after adding $n-1$ half-beams. Consider the wall $\mathbf{W}$ between rooms $\mathbf{R_i}$ and $\mathbf{R_j}$ of this hub with $n-1$ half-beams. This wall lies in the plane $\mathbf{\Pi_{ij}}$, and crossing this wall from room $\mathbf{R_i}$ will always take you to room $\mathbf{R_j}$. If we add a new half-beam (quador $\mathbf{Q_n}$) to the hub, and the half-space $\mathbf{\Pi_{ni}^+}$ corresponding to a new plane $\mathbf{\Pi_{ni}}$ trims the the side of the wall $\mathbf{W}$ in room $\mathbf{R_i}$, then by Lemma 6.5.2, there must exist plane $\mathbf{\Pi_{nj}}$, which is concurrent with planes $\mathbf{\Pi_{ij}}$ and $\mathbf{\Pi_{ni}}$, and that its half-space $\mathbf{\Pi_{nj}^+}$ trims the the side of the wall $\mathbf{W}$ in room $\mathbf{R_j}$. Hence in the new set of rooms (after adding $\mathbf{Q_n}$), either the wall $\mathbf{W}$ gets trimmed completely or gets trimmed equivalently on both sides such that crossing it from room $\mathbf{R_i}$ will always take you to room $\mathbf{R_j}$. In case the wall $\mathbf{W}$ is not trimmed completely after adding $\mathbf{Q_n}$, an edge

of the complex lies on the line of intersection of $\mathbf{\Pi_{ij}}$, $\mathbf{\Pi_{ni}}$, $\mathbf{\Pi_{nj}}$, and the neighbourhood of this edge is partitioned among the new rooms $\mathbf{R_i}$, $\mathbf{R_j}$, and $\mathbf{R_n}$. Furthermore, starting from one room, one may cross walls to travel to the other rooms of the complex, and that there is no wall separating a room from what's not inside of any of the other rooms, hence the polyhedral complex must cover the 3D space.

In the next section we discuss a representation of the trimming complex.

6.5.4 Dart representation of a trimming complex

A trimming complex has vertices, edges, walls (polygons in 3D, which each have two sides), and rooms. Brisson [69] defines a ***4-tuple (a dart)*** and represents it by (V,E,W,R), four indices to the vertex, edge incident on the vertex, wall bounded by the edge, and room bounded by the wall. He defines four types of ***switch*** (S0, S1, S2, S3) that respectively swap the vertex, or the edge, or the wall, or the room.

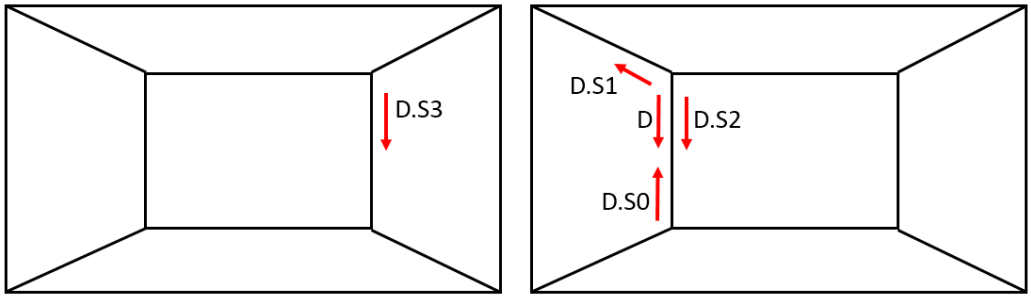


Figure 6.10: Brisson's darts and switch operators. The figure shows two rooms, a dart D on the left wall of the right room, and the result of the four switch operators on that dart.

In Brisson's representation there are two 4-tuples per edge on the same wall in a room and you can go from one to the other by an switch S0. For Random Access and Traversal (RAT), we don't need ever to distinguish between these two 4-tuples. So, we represent both of them by a single dart (Fig. 6.11).

6.5.5 Numerically robust computation of the dart representation of a trimming complex

We advocate to represent the trimming complex using ***darts*** [79, 80]. One may use CGAL's implementation of linear cell complexes [14, 81] to build this representation. CGAL claims numerical robustness of its implementation by using numbers of arbitrary precision [82]. As stated in our assumptions Sec. 6.3.2, we do not consider split-planes in our computations. Hence, some of the rooms of the trimming complex may be unbounded. To reduce the complexity arising out of that (e.g. edges that are either rays or lines), we assume that the complex is trimmed by a box bounding the hub, thereby bounding each room while maintaining connectivity of all rooms.

Additionally, we provide a numerically robust topological test in Sec. 6.7.1 to classify the intersection of three oriented planes against a fourth oriented plane. This test may be useful in computing the correct topology of a room, thereby useful in computing the correct topology of a trimming complex independent of CGAL's implementation.

6.5.6 Dart operators of a trimming complex

A dart D is the association of a room R, a wall W and a vertex V of a trimming complex. We use post-fixed notation for our dart operators and define the following base operators (Fig. 6.11) for a dart D in terms of Brisson's switch operators shown in Fig. 6.10:

- D.V: vertex of dart D
- D.W: wall of dart D
- D.R: room of dart D
- D.N = D.S0.S1 "next" around the border loop of the side of the wall
- D.S = D.S1.S2 "swing" around the vertex in a room
- D.O = D.S3.S0 "opposite" side of the wall with reversed direction to be consistent with clockwise orientation of darts on that side of the wall

We may use these base operators to construct convenient derived operators for dart D:

– D.P = D.S.N.S “previous” around the border loop of the side of the wall, note that D.P = D.S1.S0 in terms of Brisson’s switch operators.
– D.C = D.N.S, “cross” across the edge in the room, and note that D.C = D.S2.S0 as well.

To keep walking around the border of a side of a wall in a room, repeat D = D.N. To keep swinging around a vertex in a room, repeat D = D.S. To keep swinging from room to room (cw or ccw) around an edge, repeat D = D.C.O or D = D.O.C.

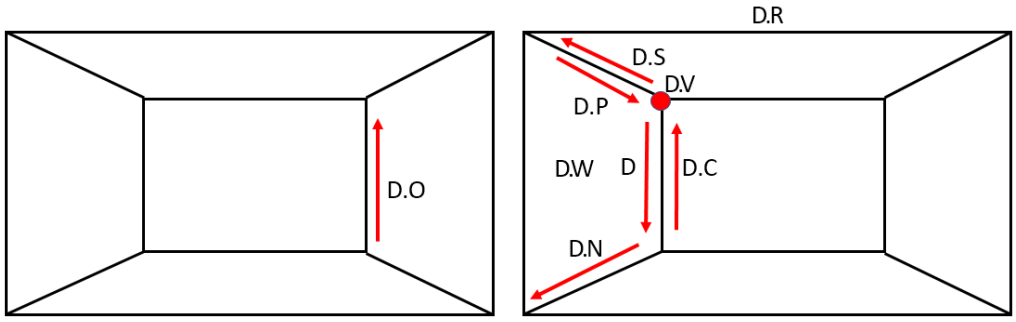


Figure 6.11: Two adjacent rooms (left and right) of a trimming complex. For simplicity we consider rooms with 6 walls. A dart D, i.e. that is the association of a room (R), wall (W) and vertex (V) of the complex is shown in the left wall D.W of the right room D.R. Vertex D.V is the top left vertex on the back wall. Darts D.N and D.P are respectively the next and previous darts in wall D.W. Dart D.S is on top wall, i.e. the next clockwise wall around D.V. Dart D.O is the dart on the other side of the wall D.W, i.e. right wall in the left room. Dart D.C is the is the cross dart, i.e. the oppositely oriented dart on the same edge and room, but on the other wall incident the edge of dart D, which in this case is the back wall.

We discussed above, computing a dart representation of the trimming complex of a hub. In the next section, we describe using this representation as a proxy to operate on the Brep of a hub.

6.6 Indirect corner operators of the Brep of a hub

Our goal is to build a corner based representation of the Brep of a hub. A corner is the association of a face and a vertex of the Brep of the hub. We consider the trimming complex of the hub, for which we have a ***dart*** based boundary representation and associated dart

operators. A dart D is the association of a Room R, Wall W and Vertex V of the trimming complex. In order to build corner operators in terms of dart operators, we associate a dart D with a corner c. Note that each corner of the hub has an associated dart but not vice versa. We define this association as follows.

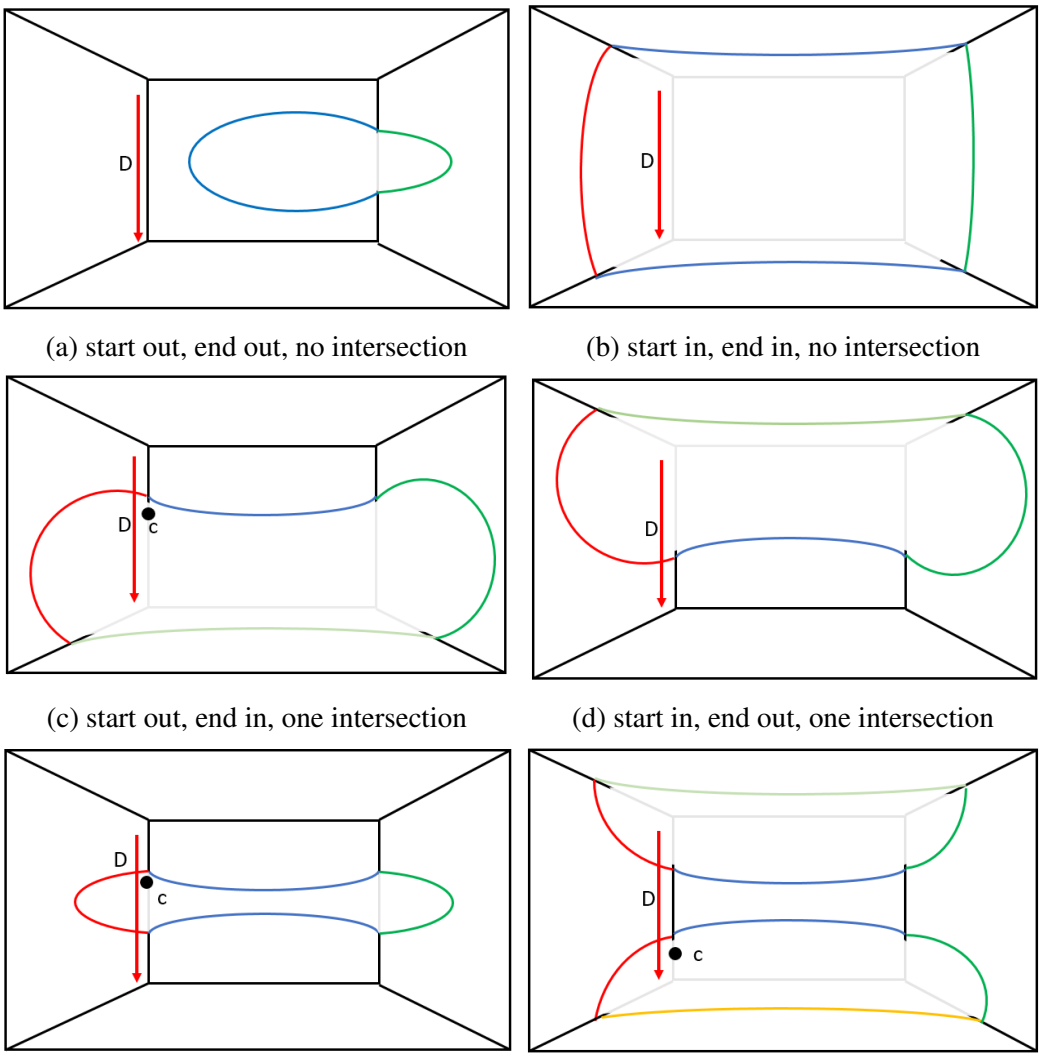
6.6.1 Computing the association of a dart with a corner of the hub

The oriented edge E associated with a dart D (i.e. edge from D.V to D.N.V) will stab the quador surface $\mathbf{Q}$ of D.R in at most two points [2]. We associate the corner in $\mathbf{Q}$ at the "first inward" stab (intersection point) of E with $\mathbf{Q}$.

We identify the first inward stab of edge E with quador surface $\mathbf{Q}$ as follows. We identify if the starting vertex (D.V) of the dart is inside or outside of the quador half-space using the predicate described in Sec. 6.7.2. We also compute the number of intersections of edge E with $\mathbf{Q}$ as using the predicate described in Sec. 6.7.3. Now the following cases may arise:

- If the number of intersections is zero then we don't have any associated corner for dart D.
- If the number of intersections is one and the starting vertex D.V is inside $\mathbf{Q}^+$, then too there is no associated corner to dart D.
- If the number of intersections is one and the starting vertex D.V is outside $\mathbf{Q}^+$, then the corner of that intersection point in $\mathbf{Q}$ is associated to dart D.
- If the number of intersections is two and starting vertex D.V is in $\mathbf{Q}^+$ then the corner of the first intersection point in $\mathbf{Q}$ is associated to dart D. The predicate described in Sec. 6.7.3 that counts the number of intersection of an edge with a quador, returns an ordered pair of parameters (t_1, t_2) (in range (0,1)) corresponding to the two intersection points.
- If the number of intersections is two and starting vertex D.V is inside $\mathbf{Q}^+$ then the corner of the second intersection point in $\mathbf{Q}$ is associated to dart D.

In Fig. 6.12, we show all the possible cases mentioned above.



(a) start out, end out, no intersection

(b) start in, end in, no intersection

(c) start out, end in, one intersection

(d) start in, end out, one intersection

(e) start out, end out, two intersection

(f) start in, end in, two intersection

Figure 6.12: Association of a corner c with a dart D in different cases. Below each subfigure we specify if the start and end of a dart are inside or outside the quador of the room of the dart, and the number of intersections of the edge of that dart with that quador. In each case the corner associated with the first inward stabbing of the edge of the dart with the quador is the associated corner of dart D.

6.6.2 Computing corner operators in terms of dart operators

Let c.D be the operator to get the dart associated with corner c and D.c be the operator to get the corner associated with dart D. Then, the corner operators of the hub are defined as

follows:

- c.v = first inward intersection of edge from c.D.V to c.D.N.V with $\mathbf{Q}$ in c.D.R.
- c.f = quador $\mathbf{Q}$ in room c.D.R
- c.s = c.D.O.C.c
- c.n = First swing twice c.D.S.S to go to the dart in the un-swinged wall of the room, then loop around the edges of that wall using next of a dart till a dart with an associated corner is found.

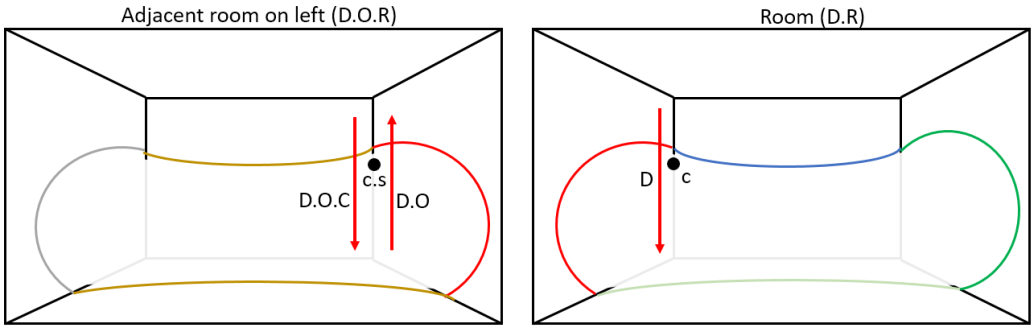
The corner operators (c.v) and (c.f) to obtain the vertex and face of a corner are trivial. In Fig. 6.13 and Fig. 6.14 we illustrate respectively how the swing and next corner operator works for the cases shown in Fig. 6.12.

6.7 Performing numerically robust topological tests using numbers in field extensions

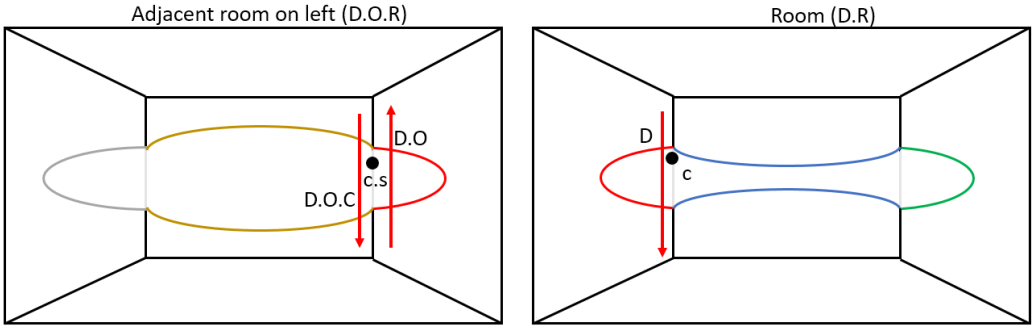
In this section we describe the topological tests that we use in our approach (Sec. 6.5, Sec. 6.6) to compute the correct topology of the Brep of a hub. We use the representations and the recursive method described in Sec. 6.2 to evaluate the signs of numbers in performing these tests.

6.7.1 Classifying a vertex of the trimming polyhedron, i.e. intersection of three trimming planes of a quador, against a fourth trimming plane of that quador

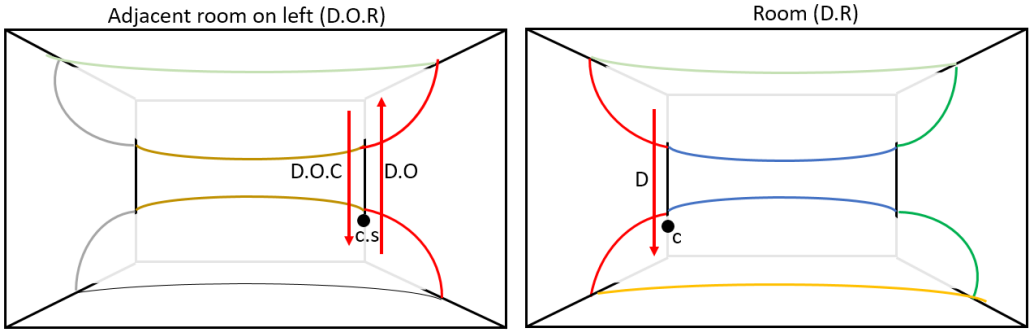
The three planes that trim $\mathbf{Q_i}$ and intersecting at P_{ijkl}, are $\mathbf{\Pi_{ji}}$,$\mathbf{\Pi_{ki}}$ and $\mathbf{\Pi_{li}}$. The coefficients of these three planes are in the field extensions $\mathbb{Q}(\sqrt{\lambda_i}, \sqrt{\lambda_j})$, $\mathbb{Q}(\sqrt{\lambda_i}, \sqrt{\lambda_k})$, and $\mathbb{Q}(\sqrt{\lambda_i}, \sqrt{\lambda_l})$ respectively (Sec. 6.3.1). Similarly, the coefficients of the plane $\mathbf{\Pi_{mi}}$ are in $\mathbb{Q}(\sqrt{\lambda_i}, \sqrt{\lambda_m})$. Then to classify P_{ijkl} against $\mathbf{\Pi_{mi}}$, we compute the determinant of a 4×4 matrix [53], where the terms are the coefficients of the four planes $\mathbf{\Pi_{ji}}$,$\mathbf{\Pi_{ki}}$, $\mathbf{\Pi_{li}}$ and $\mathbf{\Pi_{mi}}$.



(a) start out, end in, one intersection



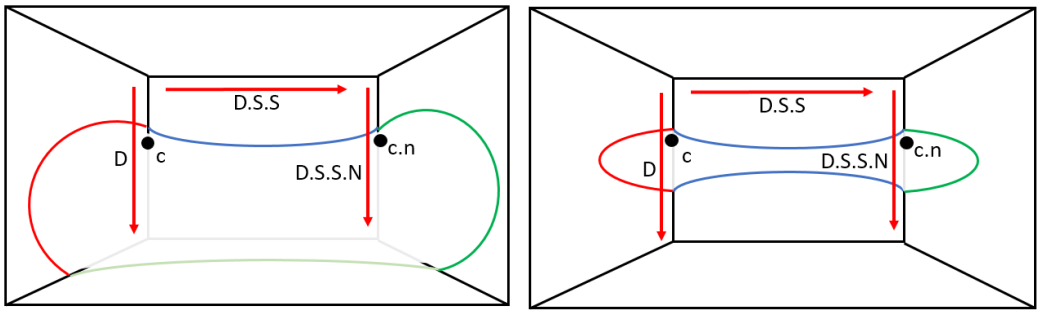
(b) start out, end out, two intersection



(c) start in, end in, two intersection

Figure 6.13: Application of the indirect swing operator c.s = c.D.O.C.c for a corner c of the hub. In each subfigure the room of the corner c is on the right and the room of its swinged corner is on the left.

Computing this determinant involves multiplying and adding these coefficients, and following our earlier discussion on arithmetic operations on numbers in field extensions, this will result in a number in $\mathbb{Q}(\sqrt{\lambda_i}, \sqrt{\lambda_j}, \sqrt{\lambda_k}, \sqrt{\lambda_l}, \sqrt{\lambda_m})$. We then evaluate if this number is 0, $+ve$, or$-ve$ to identify if P_{ijkl} is on $\mathbf{\Pi_{mi}}$, in $\mathbf{\Pi^{+}_{mi}}$ or in $\mathbf{\Pi^{-}_{mi}}$ respectively.



(a) start out, end in, one intersection

(b) start out, end out, two intersection

(c) start in, end in, two intersection

(d) additional example

Figure 6.14: The figure shows how the indirect next operator (c.n = swing twice c.D.S.S, then loop using next on darts till a dart with an associated corner is found) works for a corner c of the hub. In subfigure (d) we show an additional example to illustrate that we may have to loop over a number of edges using next of a dart to find the next of a corner.

6.7.2 Classifying a vertex of the trimming polyhedron, i.e. intersection of three trimming planes of a quador, against that quador

As mentioned above, the coefficients of the three planes intersecting at P_{ijkl} are in the field extensions $\mathbb{Q}(\sqrt{\lambda_i}, \sqrt{\lambda_j})$, $\mathbb{Q}(\sqrt{\lambda_i}, \sqrt{\lambda_k})$, and $\mathbb{Q}(\sqrt{\lambda_i}, \sqrt{\lambda_l})$ respectively. To compute the coordinates of P_{ijkl}, we solve a linear system of equations using Crammer's rule, wherein we compute ratios of determinants of 3×3 matrices of coefficients of the three intersecting planes. Which following our discussion on arithmetic operations on numbers in field extensions, will result in coordinates of P_{ijkl} to be in $\mathbb{Q}(\sqrt{\lambda_i}, \sqrt{\lambda_j}, \sqrt{\lambda_k}, \sqrt{\lambda_l})$. We then substitute these coordinates in the expression for $\mathbf{Q_i}$, which involves adding/subtracting/-multiplying these coefficients with self, each other or with other rational numbers, resulting in a number in $\mathbb{Q}(\sqrt{\lambda_i}, \sqrt{\lambda_j}, \sqrt{\lambda_k}, \sqrt{\lambda_l})$. We evaluate if this number is 0, $+ve$, or$-ve$ to

identify if P_{ijkl} is on/outside/inside of $\mathbf{Q_i}$ respectively.

6.7.3 Computing the number of intersections of an edge of the trimming polyhedron of a quador with that quador

We compute the parametric representation $P_{ijkl}+t(P_{ijkm}-P_{ijkl}) = 0$ of the line $\mathbf{E_k}(\mathbf{t})$ such that $\mathbf{E_k}(\mathbf{t = 0}) = P_{ijkl}$ and $\mathbf{E_k}(\mathbf{t = 1}) = P_{ijkm}$. Substitute the expression of this line in $\mathbf{Q_i}$ to obtain a quadratic expression in t. Let this expression be $at^2 + bt + c$. The coefficients in this expression are in $\mathbb{Q}(\sqrt{\lambda_i}, \sqrt{\lambda_j}, \sqrt{\lambda_k}, \sqrt{\lambda_l}, \sqrt{\lambda_m})$. If the sign of its discriminant $d = b^2 - 4ac$, which again is a number in $\mathbb{Q}(\sqrt{\lambda_i}, \sqrt{\lambda_j}, \sqrt{\lambda_k}, \sqrt{\lambda_l}, \sqrt{\lambda_m})$, is negative, then the line $\mathbf{E_k}(\mathbf{t})$ does not intersect $\mathbf{Q_i}$. Else if $d = 0$, then the only root is $-b/2a$, which is a number in $\mathbb{Q}(\sqrt{\lambda_i}, \sqrt{\lambda_j}, \sqrt{\lambda_k}, \sqrt{\lambda_l}, \sqrt{\lambda_m})$, and we check if it lies in $(0, 1)$ to determine the number of intersections of line segment $\mathbf{E_k}$ with $\mathbf{Q_i}$. Else $d > 0$, i.e. there are 2 roots, in which case we compute the number of roots to the right and left of the starting and end point of line segment $\mathbf{E_k}$, and from that deduce the number of intersections of $\mathbf{E_k}$ with $\mathbf{Q_i}$. The approach is as follows:

We compute the number of positive roots n_1 from the sign of the product $(p = c/a)$ and the sum $(s = -b/a)$ of roots. If p is positive, and s is negative, then both roots are negative and line segment $\mathbf{E_k}$ does not intersect $\mathbf{Q_i}$. Else, if p is positive, and s is positive then there are two positive roots, i.e. $n_1 = 2$, else if p is negative, then $n_1 = 1$. We repeat this exercise with line $\mathbf{E_k}(\mathbf{t})$ reverse parameterized with $\mathbf{E_k}(\mathbf{t = 0}) = P_{ijkm}$ and $\mathbf{E_k}(\mathbf{t = 1}) = P_{ijkl}$, i.e. as $P_{ijkm} + t(P_{ijkl} - P_{ijkm}) = 0$, substitute in $\mathbf{Q_i}$ and recompute the number of positive roots n_2. Now, from the values of n_1 and n_2, we deduce the number of intersections n of the line segment $\mathbf{E_k}$ with $\mathbf{Q_i}$ as follows:

- $n_1 = 2, n_2 = 0 \implies n = 0$
- $n_1 = 2, n_2 = 1 \implies n = 1$
- $n_1 = 2, n_2 = 2 \implies n = 2$
- $n_1 = 1, n_2 = 1 \implies n = 0$

- $n_1 = 1, n_2 = 2 \implies n = 1$

6.8 Summary of our approach to compute the correct topology of a hub

In this chapter, we presented a numerically robust approach to compute the correct topology of a hub. In our approach, we associate a convex trimming polyhedron with each surface of the hub. We conjecture that the union of these polyhedron covers the whole space and that their arrangement forms a polyhedral trimming complex. We discuss computing a numerically robust dart representation [14] of this complex and define dart operators to traverse this complex. We define the association of a corner of the Brep (Sec. 5.3) of a hub with a dart of its trimming complex, and use dart operators to traverse the Brep of the hub. To correctly compute a trimming polyhedron, and to build the association of a corner with a dart, we propose three topological tests. We prove that that these tests can be performed by computing the signs of closed-form algebraic expressions, each involving only rational numbers and up to five non-nested square roots. We describe an approach to evaluate the sign of such an expression using only integers and only the standard arithmetic operations.

As we decompose a lattice of quador beams into an assembly of pairwise interior-disjoint hubs (Ch. 4), the correct topology of the Brep of a lattice can be computed trivially from that of its hubs. Though one may accelerate it through distributed computing, it may still be computationally unviable for enormous lattices with billions to trillions of hubs. In the next chapter, we propose a class of lattices called ***Steady Lattices*** which consists of patterns of hubs, with hubs of a pattern related by similarity transforms. Hence, one may compute the correct connectivity and numerically approximate geometry of the Brep of one of the hubs of a pattern and obtain the same of all the other hubs of that pattern by simple similarity transforms.

CHAPTER 7
STEADY LATTICES

In the previous chapters of this thesis, we addressed our first challenge that we talked about in the introduction, that is of designing and modeling beams with curved profiles and of computing their surfaces reliably. We now address the second challenge of precisely modeling highly complex lattices with profiled beams and of efficiently computing their mass properties.

Our goal is to support design and processing of large and complex lattices like the one shown in Fig. 7.1. This requires representing the lattice and therefore its hubs in a form that is easy to compute, compact to store and is amenable to fast and robust queries. We propose such a representation for a class of lattices, that we call ***steady lattices***.

In this chapter, we first go through prior art on lattice structures, we then define steady lattices, followed by efficient computation of a specific set of balls and beams of a steady lattice. Thereafter, we discuss special properties of steady lattices that helps in efficient processing of a large and complex steady lattice. Then, in the last section of this chapter, we show examples of steady lattices and report the performance of computing the parameters defining the balls and beams of a steady lattice.

7.1 Prior art on lattice microstructures

We divide prior art into (1) Procedural models, (2) Regular lattices, (3) Conformal lattices, and (4) Irregular models.

7.1.1 Procedural Models

Our approach of representing the lattice by a program builds on earlier and more general procedural modeling paradigms [83]. For example, the MAMOUR system [84, 85] en-

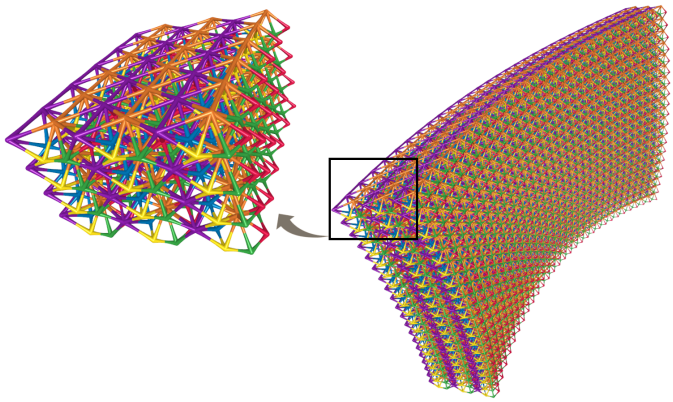


Figure 7.1: A large and complex lattice with more than 50,000 quador beams.

abled to define patterns of repeated features (such as a string of uniformly spaced rivet holes around a face of a CAD model), to parameterize them, and to program parameter expressions in terms of geometric measures. Application of procedural modeling to repeated patterns and lattice structures are numerous.

The ABCSG system [86] supported a text-based representation of a scene-graph that combines solid primitives with operators (assembly, Boolean, Affine Transformation, Repetition, and Recursion) that can be edited with a keyboard or a GUI. The OCTOR system [87] provided effective tools for using the GUI to select sub-structures in such iterative or recursive patterns, so as to edit them or to specify exceptions at which instances do not follow the pattern.

Procedural methods that define geometry and coordinate systems using analytic implicit functions [88] have also been used to generate lattices.

7.1.2 Regular lattices

Regular lattices are the most common type of lattice structures used in the design of mechanical objects [89].

They consist of copies of a cell template arranged on a regular grid. These types of lattices have also been used to design meta materials with microstructures, such as architected materials [6]. Interactive modeling and design is performed by orienting the grid,

specifying cell templates, and selecting cell sizes and cell instance counts.

Solid modeling operations, such as extrude and sweep, have been applied to the design of lattice cell layouts [7].

R-Functions [88] have also been used to model regular lattices and their variations, that for example include cylindrical and spherical patterns.

While the simplicity of these structures enables an efficient representation, the regularity of such structures constrains the kinds of lattices that can be modeled with this approach.

7.1.3 Conformal Lattices

Conformal lattices [11, 90] address the above limitation by requiring that edges or vertices of one of the faces of each cell of a base layer be aligned with iso-parametric curves of a surface. Additional layers of cells may be added by offsetting the base layer of conformal cells, or by interpolating between two base conformal layers lying on different surfaces [11, 90].

Recent work extends the conformal lattice patterning approach to the volumetric domain [91]. Here, all cells are aligned according to iso-parametric curves of trivariate B-Splines. In this approach, the geometry of the cell template is deformed according to the trivariate mapping.

The cell layout [7] may also be defined by a volumetric mesh, such as a tetrahedral or hexahedral mesh. Cell templates are deformed to match the shape of the corresponding volume mesh elements. A variant has been demonstrated in the Plato software system developed by Sandia Labs [92].

Alternatively, the vertices and edges of the volume mesh may themselves be used to define the lattice structure [7].

Implicit functions, including distance fields have also been used to define lattices [93, 7] and heterogeneous materials [94].

Warped lattice structures were created in [95] by morphing a 3D rectangular grid ac-

cording to a warping function. The warping function was computed from an optimization procedure. Lattice templates were then morphed into each grid cell to create a continuous lattice structure.

Skin lattices that follow a prescribed surface may also be created [96] by trimming voxelized representation of functionally graded lattices.

These approaches broaden considerably the design space of regular lattices. Most of them require storing evaluated models of the supporting representations (such as the topology and geometry of tetrahedral meshes, the control points and knots of the control grids of splines, or the coefficients of the basis functions). The associated storage cost makes memory access expensive when processing lattices with billions of elements. More importantly, the irregularity of the patterns that they support complicates the task of identifying which cells lie near a given query (point, line, or plane) and makes geometric and integral queries expensive. Finally, we conjecture that the lack of steadiness of the patterns may often result in non-monotonic gradations of mechanical properties, which, we believe can make analysis and optimization more challenging, and ultimately may reduce the performance of the final product.

7.1.4 Irregular Models

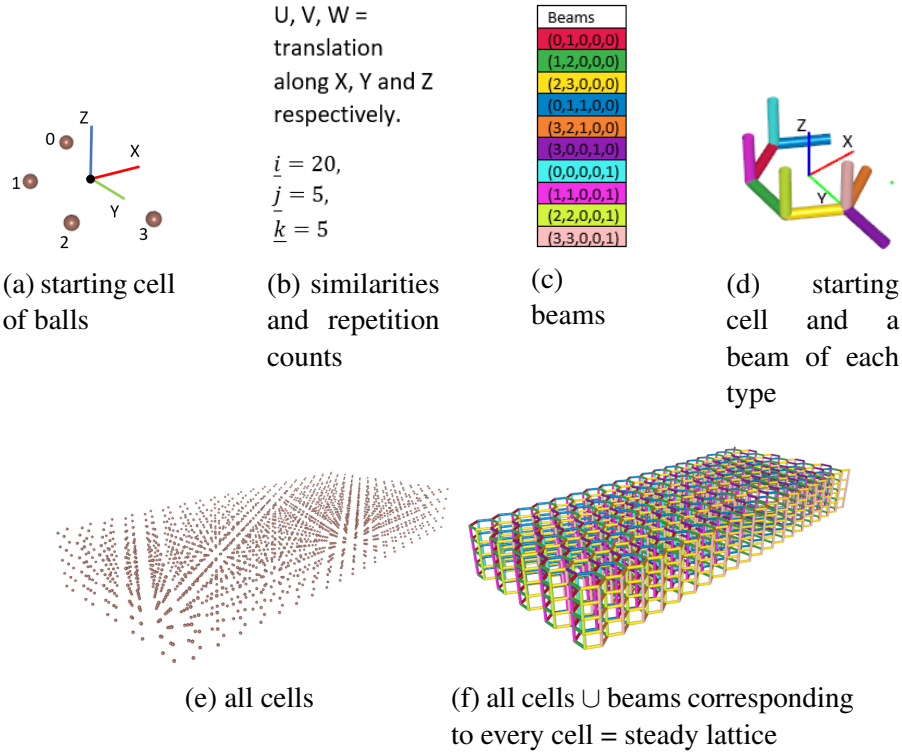
A large body of literature focuses on procedural generation of non-periodic structures [97, 98]. These non-periodic lattices make it simple to grade the geometry of the microstructure, but do not offer precise control to define the geometry and topology of the microstructure. For example, procedural Voronoi Foams [99, 100] is an example of a non-fully evaluated lattice representation. Such lattices are modeled similarly to Voronoi-based, procedural, cellular textures, which are very efficient to evaluate and store. However, the method is restricted to modeling stochastic, open-cell foams.

7.2 Definition of a steady lattice

We propose a Parametric Program-Representation (PPR) of periodic lattices ([3]) with repeating patterns of elements (balls and beams) in three different directions, wherein we represent a lattice by a short program with a selected set of exposed control parameters that may be used to adjust (and possibly optimize for some application) the overall shape of the lattice, its repetition count in each direction, its microstructure, and its gradation. We call these lattices ***Steady Lattices*** and define them as follows:

The balls of a steady lattice are organized into an $\underline{i} \times \underline{j} \times \underline{k}$ array of ***cells*** (a small set of balls), and that there are three similarities, $(\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W})$, such that a cell $\mathbf{C}[i, j, k]$ is a similarity transform of a starting cell $\mathbf{C}[0, 0, 0]$, i.e. $\mathbf{C}[i, j, k] = \boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \boldsymbol{U}^i \cdot \mathbf{C}[0, 0, 0]$, which implies that the ball n in cell $\mathbf{C}[i, j, k]$, denoted as $\mathbf{N}[i, j, k, n]$, is given by $\boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \boldsymbol{U}^i \cdot \mathbf{N}[0, 0, 0, n]$. A beam defined by $\mathbf{B}(s, e, d_i, d_j, d_k)$ means that the lattice contains all beams that each connects ball $\mathbf{N}[i, j, k, s]$ to ball $\mathbf{N}[i + d_i, j + d_j, k + d_k, e]$ for all valid triplets (i, j, k). A set of solids $\mathbf{X_i}$ is a *steady pattern* when there exists a similarity $\boldsymbol{S}$ such that, for each valid index i, $\mathbf{X_i} = \boldsymbol{S}^k \cdot \mathbf{X_0}$, where $\mathbf{X_0}$ is the first solid in the pattern. We prove in Sec. 7.4 that a steady lattice consists of ***steady rows*** of hubs, wherein the hubs in each row form a steady pattern along the k direction. Figure 7.2 shows an example of a steady lattice and illustrates the associated terminology.

Thus to specify a steady lattice, we need to specify a starting cell of balls $\mathbf{C}[0, 0, 0]$, a set of beams, each specified in form $\mathbf{B}(s, e, d_i, d_j, d_k)$, three similarity transforms $(\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W})$, and three corresponding repetition counts $(\underline{i}, \underline{j}, \underline{k})$. Besides, a cone beam being implicitly defined from the position of its two balls (Ch. 2), steady lattices with cone beams do not require any addition parameter to specify the shape of beams, but lattices with quador beams do. For lattices with quador beams, we specify a single parameter x for the whole lattice, to compute the "thickening parameter" (Sec. 3.2.3) $t_x = t_{min} + x(t_{max} - t_{min})$ for each of the quador beams of the lattice.



(a) starting cell of balls

(b) similarities and repetition counts

(c) beams

(d) starting cell and a beam of each type

(e) all cells

(f) all cells ∪ beams corresponding to every cell = steady lattice

Figure 7.2: Example illustrating terminology associated with a steady lattice.

Note that the lattice shown in Fig. 7.1 is a Steady Lattice, defined by specifying a starting cell with just 6 balls, 36 beams and three similarity transforms.

7.3 Computing a cell (set of balls) of a steady lattice

The overall shape and gradation of a steady lattice is controlled by three similarity transformations. This deliberate design choice ensures that the gradation in each direction is regular (i.e., mathematically steady), and that each cell can be evaluated ***directly***, without iterations. To compute the balls in cell $\mathbf{C}[i,j,k] = \boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \boldsymbol{U}^i \cdot \mathbf{C}[0,0,0]$ in a steady lattice, we need to compute the powers $(\boldsymbol{U}^i, \boldsymbol{V}^j, \boldsymbol{W}^k)$ of similarity transforms. To compute the power of a similarity, e.g. $\boldsymbol{W}^t$, where t is a real valued exponent, we decompose $\boldsymbol{W}$ it into the canonical commutative product $\boldsymbol{R} \cdot \boldsymbol{T}$, where $\boldsymbol{R}$ and $\boldsymbol{T}$ are primitive transformations (translation, rotation and scaling), such that $\boldsymbol{W}^t = \boldsymbol{R}^t \cdot \boldsymbol{T}^t$.

We distinguish two cases, when $m_w = 1$, $\boldsymbol{W}$ is a rigid body transformation. Then, $\boldsymbol{R}$

is a rotation and $\boldsymbol{T}$ is a translation by a vector parallel to the axis of $\boldsymbol{R}$. In this case, as t varies, $\boldsymbol{W}^t$ defines a ***screw motion*** and point $\boldsymbol{W}^t \cdot P_0$ traces a ***helix*** (See [101] for the computation of $\boldsymbol{W}^t$.)

When $m_w \neq 1$, $\boldsymbol{W}$ is a similarity. Then, $\boldsymbol{R}$ is a rotation and $\boldsymbol{T}$ a dilation about a fixed point F on the axis of $\boldsymbol{R}$. In this case, as t varies, $\boldsymbol{W}^t$ defines a ***swirl motion*** and point $\boldsymbol{W}^t \cdot P_0$ traces a ***concho-spiral*** [102].

The precise expressions for these two motions are given below. The notations used in these expressions are given in Appendix A and their derivation is given in Appendix C. We evaluate them for uniformly spaced values of t to produce steady ***screw patterns*** or steady ***swirl patterns***.

$$\boldsymbol{W}^t \cdot P_0 := F + (td)\overrightarrow{N} + (\overrightarrow{FP_0})^\circ(t\alpha, \overrightarrow{N}) \tag{7.1}$$

$$\boldsymbol{W}^t \cdot P_0 := F + m_w^t(\overrightarrow{FP_0})^\circ(t\alpha, \overrightarrow{N}) \tag{7.2}$$

Here, F is the fixed point, $\overrightarrow{N}$ is the unit vector along the axis of rotation, α is the angle of rotation, d is the distance of translation along $\overrightarrow{N}$, and m_w is scaling. Furthermore, $\overrightarrow{FP_0}^\circ(\alpha, \overrightarrow{N})$ is $\overrightarrow{FP_0}$ rotated by α around $\overrightarrow{N}$. Note that in special cases, the first expression may reduce to pure translation ($\alpha = 0$) or pure rotation ($d = 0$) and the second expression may reduce to pure scaling ($\alpha = 0$).

To compute the balls of cell $\mathbf{C}[i, j, k]$, we transform each ball of cell $\mathbf{C}[0, 0, 0]$, by the appropriate similarity. To do so, we precompute the canonical decomposition of each of the three similarities $\boldsymbol{U}$, $\boldsymbol{V}$ and $\boldsymbol{W}$ of the steady lattice. Then, we compute the center and radius of ball $\mathbf{N}[i, j, k, n]$ by transforming the center C_n and radius r_n of the corresponding ball in the starting cell as follows, assuming that m_u, m_v and m_w are the scaling factors of $\boldsymbol{U}$, $\boldsymbol{V}$ and $\boldsymbol{W}$ respectively.

$$\mathbf{N}[i,j,k,n] = Ball(\boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \boldsymbol{U}^i \cdot C_n,\ m_w^k m_v^j m_u^i r_n) \tag{7.3}$$

7.4 Steady rows of hubs in a steady lattice

In a Steady Lattice each cell can be expressed as $\mathbf{C}[i,j,k] = \boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \boldsymbol{U}^i \cdot \mathbf{C}_0$, where $\mathbf{C}_0 = \mathbf{C}[0,0,0]$. We consider the patterns of cells and therefore the patterns of each of its balls, in the i, j, and k directions. We prove that they are steady patterns. We also prove that each beam belongs to a steady pattern of beams in the k direction. We conclude that, each hub belongs to a steady pattern of hubs in the k direction.

7.4.1 Steady patterns of cells and balls

Consider the following two steps, through which we reformulate $\boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \boldsymbol{U}^i \cdot \mathbf{C}_0$:

$$\begin{aligned} \mathbf{C}[i,j,k] &= \boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \boldsymbol{U}^i \cdot \mathbf{C}_0 \\ &= (\boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \boldsymbol{U} \cdot \boldsymbol{V}^{-j} \cdot \boldsymbol{W}^{-k})^i \cdot (\boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \mathbf{C}_0) \\ &= \boldsymbol{M}^i \cdot \mathbf{C}[0,j,k], \text{where } \boldsymbol{M} \text{ is a similarity.} \end{aligned} \tag{7.4}$$

where, $\boldsymbol{M} = \boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \boldsymbol{U} \cdot \boldsymbol{V}^{-j} \cdot \boldsymbol{W}^{-k}$ and $\mathbf{C}[0,j,k] = \boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \mathbf{C}_0$.

Eq. 7.4 shows that the set of cells $\mathbf{C}_i$ for a fixed index pair (j,k) forms a steady pattern with the starting cell $\mathbf{C}[0,j,k]$.

Similarly, we prove below that the set of cells along $\mathbf{C}_j$ for a fixed index pair (i,k) forms a steady pattern with the starting cell $\mathbf{C}[i,0,k]$.

$$\begin{aligned}\mathbf{C}[i,j,k] &= \boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \boldsymbol{U}^i \cdot \mathbf{C}_0 \\ &= (\boldsymbol{W}^k \cdot \boldsymbol{V} \cdot \boldsymbol{W}^{-k})^j \cdot (\boldsymbol{W}^k \cdot \boldsymbol{U}^i \cdot \mathbf{C}_0) \\ &= \boldsymbol{M}^j \cdot \mathbf{C}[i,0,k]\end{aligned} \tag{7.5}$$

where, $\boldsymbol{M} = \boldsymbol{W}^k \cdot \boldsymbol{V} \cdot \boldsymbol{W}^{-k}$ and $\mathbf{C}[i,0,k] = \boldsymbol{W}^k \cdot \boldsymbol{U}^i \cdot \mathbf{C}_0$.

Finally, we prove below that the set of cells along $\mathbf{C}_k$ for a fixed index pair (i,j) forms a steady pattern with the starting cell $\mathbf{C}[i,j,0]$.

$$\begin{aligned}\mathbf{C}[i,j,k] &= \boldsymbol{W}^k \cdot \boldsymbol{V}^j \cdot \boldsymbol{U}^i \cdot \mathbf{C}_0 \\ &= (\boldsymbol{W}^k) \cdot (\boldsymbol{V}^j \cdot \boldsymbol{U}^i \cdot \mathbf{C}_0) \\ &= \boldsymbol{M}^k \cdot \mathbf{C}[i,j,0]\end{aligned} \tag{7.6}$$

where, $\boldsymbol{M} = \boldsymbol{W}$ and $\mathbf{C}[i,j,0] = \boldsymbol{V}^j \cdot \boldsymbol{U}^i \cdot \mathbf{C}_0$.

7.4.2 Steady patterns of beams

In this subsection, we prove that given a beam $\mathbf{B}[s,e,d_i,d_j,d_k]$ connecting ball $\mathbf{N}[i,j,k,s]$ to ball $\mathbf{N}[i+d_i, j+d_j, k+d_k, e]$, the set $\mathbf{B}_k$ of its instances in cells $\mathbf{C}_k$ for a fixed index pair (i,j) forms a steady pattern. We already proved above the steadiness of the set of cells $\mathbf{C}_k$, hence for balls $\mathbf{N}[i,j,k,s]$ and $\mathbf{N}[i+d_i, j+d_j, k+d_k, e]$ in these cells:

$$\begin{aligned}\mathbf{N}[i,j,k,s] &= \boldsymbol{W}^k \cdot \mathbf{N}[i,j,0,s] \\ \mathbf{N}[i+d_i, j+d_j, k+d_k, e] &= \boldsymbol{W}^k \cdot \mathbf{N}[i+d_i, j+d_j, d_k, e]\end{aligned} \tag{7.7}$$

As both the balls defining the set of beams mentioned above, follow the steady pattern defined by $\boldsymbol{W}$, the set of beams $\boldsymbol{B}_k$ forms a steady pattern.

7.4.3 Steady patterns of hubs

In Sec. 4.1 we defined a *hub* as the union of a ball and its half-beams. In the sub-sections above, we proved that a steady lattice consists of steady pattern of cells (of balls) $\mathbf{C}_k$ and of beams $\mathbf{B}_k$ for a fixed index pair (i, j). Moreover, both for the cells and for the beams, the steady pattern is defined by similarity $\boldsymbol{W}$, hence in a steady lattice, the set of hubs $\mathbf{H}_k$ for a fixed set of index pair (i, j) forms a steady pattern, i.e. a hub $\mathbf{H}[i, j, k, n]$ at ball n in cell $\mathbf{C}[i, j, k]$ is given by $\boldsymbol{W}^k \cdot \mathbf{H}[i, j, 0, n]$ for a fixed index pair (i, j).

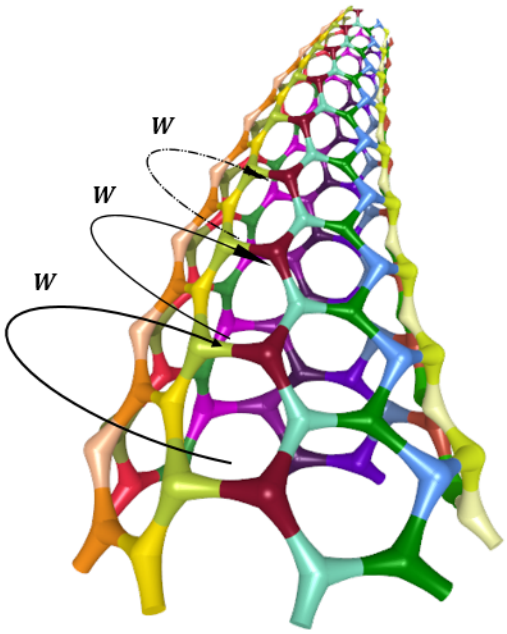


Figure 7.3: A steady lattice has steady rows of hubs along the k direction. In the above image steady rows of hubs are colored differently and hubs of a row have the same color. In the example above, each hub in a steady row is a similarity transform of the previous hub in that row by the similarity transform $\boldsymbol{W}$.

7.5 Examples of steady lattices

Fig. 7.4 shows a steady lattice and the small program defining that lattice. With just a few lines of code we have created a reasonably complex lattice that bends and grades. Notice that in this example, some of the parameters (namely "ringRadius", "ballRadius"

and ”scale”) defining the lattice are exposed to control and possibly optimize the lattice. Fig. 7.5 shows instances of a more complex steady lattice. Each instance is created by changing one or more of the exposed control parameters of that lattice.

```
class Params { float ringRadius=100, ballRadius=5, scale=1.2; }
..

float R=params.ringRadius, r=params.ballRadius, s=params.scale;
Pt p = Pt(1000,0,0);

Node a = addNode(Pt(R+p.x,0,0), r);

addBeam(a,a, 1,0,0); // Red
addBeam(a,a, 0,1,0); // Green
addBeam(a,a, 0,0,1); // Yellow

int i=7, j=20, k=50;
setU(i, Translation(20,0,0).Scale(s,p));
setV(j, Rotation(2*PI/j, Vec(0,0,1), p));
setW(k, Translation(0,100,0).Scale(s).Rotate(-4*PI/k, Vec(0,1,0)));

closeDirectionV();
```

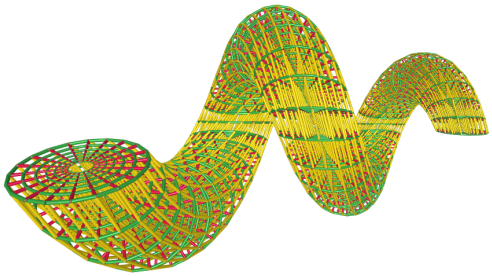


Figure 7.4: A program defining a steady lattice and the corresponding lattice. The class, Params, exposes parameters that may be used by an engineer or by an optimization program to tweak the lattice.

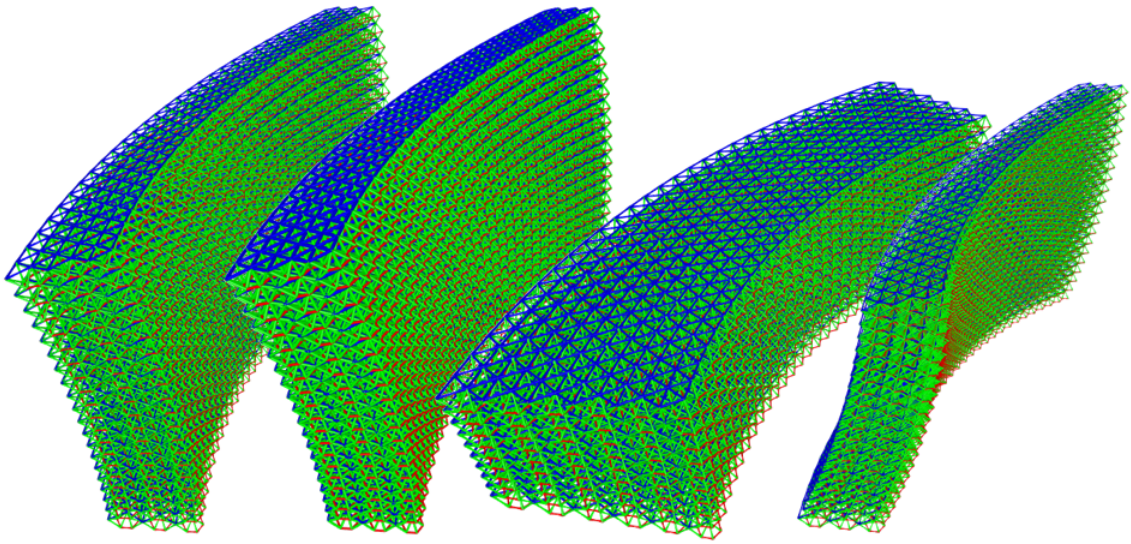
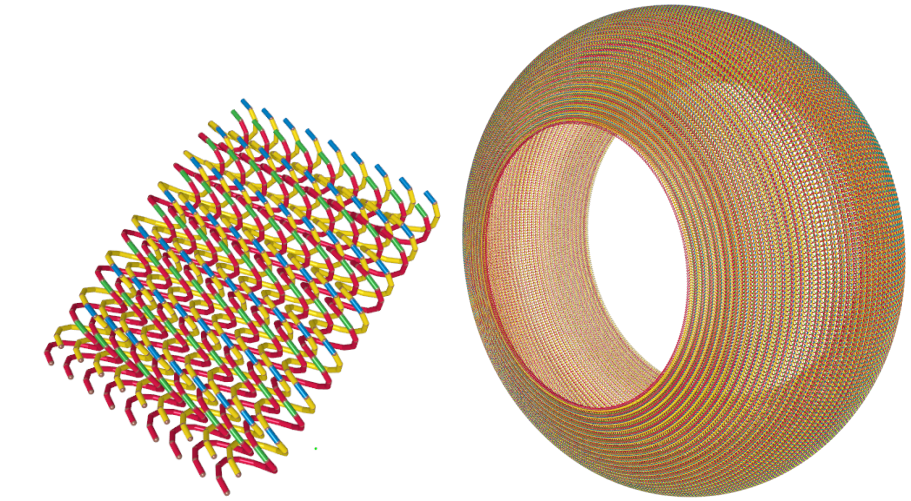


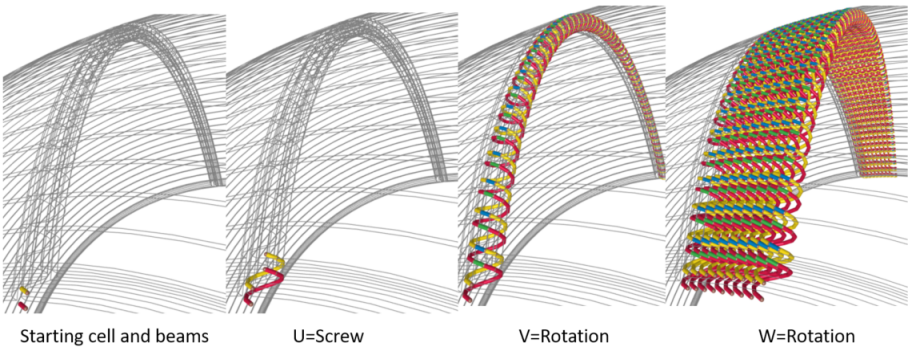
Figure 7.5: Instances of a steady lattice with quador beams, obtained by changing one or more of its control parameters. From left, the first instance has thin beams, the second instance has thick beams, the third instance has different repetition counts than the first two, and the fourth instance has a different similarity transform along its height than the previous three instances.

Another example of a complex steady lattice is shown in Fig. 7.6. This lattice is inspired by the “Superealastic Tire” design by NASA [103]. A full scale model (Fig. 7.7-right) of such a steady lattice was designed and printed by Siemens Corporate research, one

of our collaborators in this research.

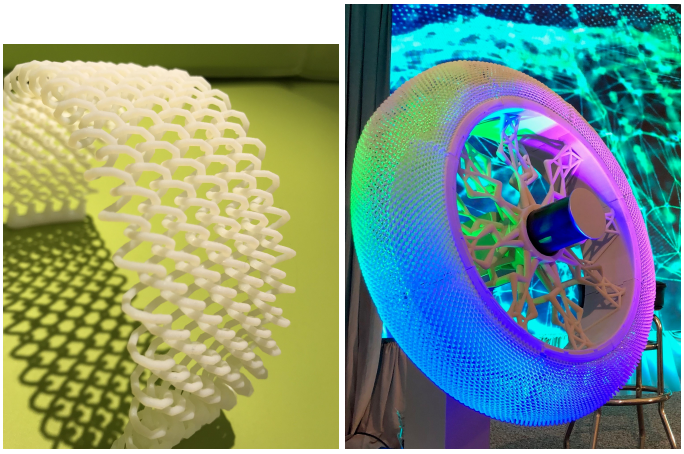


(a) Weave structure (b) Spring tire



(c) Spring tire construction

Figure 7.6: Construction of a spring-tire lattice.



(a) Reduced scale model (b) Full scale model

Figure 7.7: 3D printed models of a spring tire lattice.

7.6 Experimental validation of computing balls and beams of a steady lattices

Steady lattices are implicitly defined lattices, i.e. the balls and beams of the lattice never needs to be evaluated and stored in its entirety. Instead, they are evaluated and queried on demand for a Region-of-Interest (RoI). To report the performance (Table 7.1) of computing the balls and beams of a steady lattice, we consider the lattice of Fig. 7.8. For each beam, we compute the centers and radii of its two balls and the centers and radii of the two circular edges of the beam (truncated cone). We perform tests with increasing number of repetitions in each direction and for each test we average the timings of three runs. Due to direct computation of balls (Sec. 7.3) and simple implicit computation of beams 3.2, we are able to process on an average 2.5 million beams per second on a (3.6GHZ, Quad Core, i7-7700 CPU) without any customized parallel processing.).

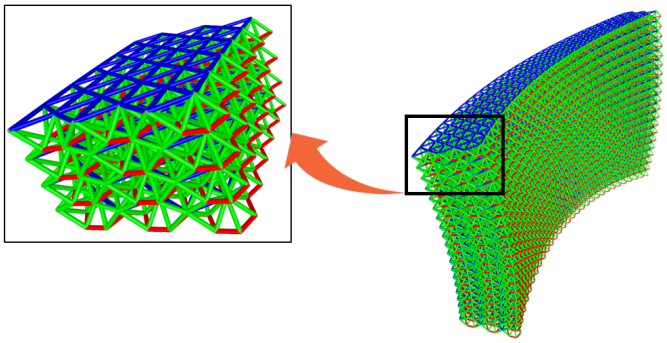


Figure 7.8: A steady lattice with cone beams.

Table 7.1: Performance of computing the balls and beams of the Honeycomb-Dual lattice of Fig. 1.1.

# Repetions in each direction	# Beams	Time taken (ms) - average of three runs	# Beams processed per Sec.
10	36000	16.666666	2160000
20	288000	114.666664	2511627
50	4500000	1720	2616279
100	36000000	14430.667	2494687

In this chapter we described steady lattices, a new class of lattices that are suitable for designing and processing large and complex lattices. We may use steady lattices to design the interior of a 3D printed part. A common objective of designing the interior of a 3D printed part with lattice microstructures is to increase the strength to weight ratio of the part, i.e. to reduce the overall weight of the part while maintaining required strength. Similarly for other applications (e.g. heat transfer, adsorption), it may be required to increase the surface area to volume ratio of the lattice. To achieve these objectives, it may require computing the mass properties, such as the surface area or the volume of the lattice microstructure. It may not be viable to compute mass properties of a large lattice by iterating over its billions of elements. In the next chapter we discuss efficiently computing the mass properties of a steady lattice.

CHAPTER 8
ACCELERATING THE COMPUTATION OF THE MASS PROPERTIES OF A STEADY LATTICE

In general, optimizing a lattice for a certain application, may include constraints such as, minimizing the overall weight, or achieve a certain distribution of weight, or maximize the total surface area of the lattice. This necessitate, computing the mass properties, such as the volume or surface area of a lattice. Computing these properties for a large lattice is complicated by two challenges, one, that it is unviable to compute these properties by iterating over all the elements in a large lattice, and two, that it is difficult to compute these properties due to geometric complexity of the intersections between the balls and beams of a lattice. In this chapter we address these challenges to closely and efficiently estimate the mass properties of a steady lattice.

8.1 Closed-form expressions for mass properties of a steady lattice

The first of these two challenges is partially addressed by defining the lattice as a steady lattice. Given that a steady lattice consists of steady rows of hubs, we exploit this property to provide closed-form expressions to compute the mass properties of a steady lattice. These closed-form expressions reduces the computational complexity from $O(\underline{i} \times \underline{j} \times \underline{k})$ to $O(\underline{i} \times \underline{j})$. In other words, for an $n \times n \times n$ steady lattice, if the time taken to compute mass properties by brute force is t, then using the closed-form expression it will take only $t^{2/3}$. We derive the closed-form expressions for computing specific integral properties, namely *surface area, volume, center of mass, and spherical inertia* of a steady lattice as follows:

8.1.1 Surface area of a steady lattice

Let a_0 be the total surface area of all the hubs (without the end caps of half-beams) in the $k = 0$ layer of cells and let a_k denote the surface area of all the hubs in k^{th} layer of cells.

If $\boldsymbol{W}^k$ is a screw pattern (Sec. 7.3), each *hub* of the $k = 0$ layer undergoes a rigid transformation, hence the total surface area a_L of the lattice is simply $\underline{k}a_0$.

If $\boldsymbol{W}^k$ is a swirl pattern (Sec. 7.3), a *hub* of the $k = 0$ layer is scaled by m_w^{2k}, hence the total surface area of the lattice is:

$$a_L = \sum a_k = a_0 \sum (m_w^2)^k = \frac{a_0(m_w^{2\underline{k}} - 1)}{(m_w^2 - 1)} \tag{8.1}$$

In the equation above and onwards in this section, we will use $\sum$ to denote $\sum_{k=0}^{\underline{k}-1}$ for simplicity.

8.1.2 Volume of a steady lattice

Let v_0 be the total volume of all the hubs in the $k = 0$ layer of cells and v_k be the volume of all the hubs in the k^{th} layer of cells.

If $\boldsymbol{W}^k$ is a screw pattern (Sec. 7.3), the total volume v_L of the lattice is $\underline{k}v_0$.

If $\boldsymbol{W}^k$ is a swirl pattern (Sec. 7.3), the volume of the lattice is:

$$v_L = \sum v_k = v_0 \sum (m_w^3)^k = \frac{v_0(m_w^{3\underline{k}} - 1)}{(m_w^3 - 1)} \tag{8.2}$$

8.1.3 Centroid of a steady lattice

Let G_0 be the *centroid* (center of mass) of all the hubs in the $k = 0$ layer of cells. Then, assuming uniform material density through out the lattice, the centroid G_L of the whole lattice can be computed as the weighted average of the centroids of all the k layers of cells:

$$G_L = \frac{\sum v_k \boldsymbol{W}^k \cdot G_0}{v_L} \tag{8.3}$$

Swirl pattern

In case of a swirl pattern (Sec. 7.3), the expression for centroid is:

$$G_L = \frac{v_0 \sum (m_w^3)^k \boldsymbol{W}^k \cdot G_0}{v_0((m_w^3)^{\underline{k}} - 1)/(m_w^3 - 1)} \tag{8.4}$$

Now, from Eq. 7.2:

$$\boldsymbol{W}^k \cdot G_0 = F + m_w^k (\overrightarrow{FG_0})^\circ (k\alpha, \overrightarrow{N}) \tag{8.5}$$

Let $\overrightarrow{V} = \overrightarrow{FG_0}$, $\overrightarrow{V_1} = (\overrightarrow{V}.\overrightarrow{N})\overrightarrow{N}$, $\overrightarrow{V_2} = \overrightarrow{V} - \overrightarrow{V_1}$, $c_k = \cos(k\alpha)$ and $s_k = \sin(k\alpha)$, then:

$$\begin{aligned} \overrightarrow{V}^\circ (k\alpha, \overrightarrow{N}) &= (\overrightarrow{V_1} + \overrightarrow{V_2})^\circ (k\alpha, \overrightarrow{N}) \\ &= \overrightarrow{V_1} + \overrightarrow{V_2}^\circ (k\alpha, \overrightarrow{N}) \\ &= \overrightarrow{V_1} + c_k \overrightarrow{V_2} + s_k \overrightarrow{V_2}^\circ (\overrightarrow{N}) \end{aligned} \tag{8.6}$$

Using the above expression in Eq. 8.4, centroid G_L is:

$$G_L = \frac{\begin{aligned}(\textstyle\sum m_w^{3k})F + (\sum m_w^{4k})\overrightarrow{V_1} + (\sum m_w^{4k} c_k)\overrightarrow{V_2} \\ + (\textstyle\sum m_w^{4k} s_k)\overrightarrow{V_2}^\circ (\overrightarrow{N})\end{aligned}}{(m_w^{3\underline{k}} - 1)/(m_w^3 - 1)} \tag{8.7}$$

While other summations in the above expression are trivial, the summations with trigono-

metric terms can be computed by substituting $a = m_w^4$ in the following expressions [104]:

$$\begin{aligned} \sum a^k c_k &= \frac{1 - ac_1 - a^{\underline{k}} c_{\underline{k}} + a^{\underline{k}+1} c_{\underline{k}-1}}{1 + a^2 - 2ac_1} \\ \sum a^k s_k &= \frac{as_1 - a^{\underline{k}} s_{\underline{k}} + a^{\underline{k}+1} s_{\underline{k}-1}}{1 + a^2 - 2ac_1} \end{aligned} \tag{8.8}$$

Screw pattern

In case of screw pattern (Sec. 7.3), the volume of each k layer remains constant, thus:

$$G_L = \frac{v_0 \sum \boldsymbol{W}^k \cdot G_0}{\underline{k} v_0} \tag{8.9}$$

From Eq. 7.2:

$$\boldsymbol{W}^k \cdot G_0 = F + (kd)\overrightarrow{N} + (\overrightarrow{FG_0})^\circ(k\alpha, \overrightarrow{N}) \tag{8.10}$$

Similar to the derivation of centroid in the swirl case, the centroid for the screw case can be expressed as:

$$G_L = \frac{\begin{array}{r} \underline{k}F + \underline{k}\overrightarrow{V_1} + (d\sum k)\overrightarrow{N} + (\sum c_{\underline{k}})\overrightarrow{V_2} \\ + (\sum s_k)\overrightarrow{V_2}^\circ(\overrightarrow{N}) \end{array}}{\underline{k}} \tag{8.11}$$

The summations with trigonometric terms can be computed by substituting $a = 1$ in Eq. 8.8.

8.1.4 Spherical moment of inertia of a steady lattice

The spherical moment of inertia, or simply "***spherical inertia***", of a body B about a fixed point G is $i_G = \int_B r^2 dm$, where r is the distance of the infinitesimal mass dm from G and body's total mass is $m = \int_B dm$. For example, for a ball of uniform density ρ, center G and radius r, $i_G = \rho(4/5)\pi r^5$ and $m = \rho(4/3)\pi r^3$. The spherical inertia of the body about a different point P can be transferred from G using $i_P = i_G + m|\overrightarrow{PG}|^2$.

We compute the spherical inertia of a steady lattice L about its centroid G_L as follows.

We assume, that we can compute the mass m, the centroid G, and the spherical inertia i for each hub in the $k = 0$ layer of hubs, about their respective centroid. We can then compute the mass m_0, centroid G_0, and spherical inertia i_0 of the entire $k = 0$ layer about its centroid G_0. The spherical inertia i_0 is computed by using the transfer formula given above and summing the results. We compute the center of mass, G_L of the lattice and then compute the spherical inertia i_L of the whole lattice about G_L, by combining the inertia of all the k layers as follows:

$$i_L = \sum (i_k + m_k |\overrightarrow{G_L G_k}|^2) \tag{8.12}$$

where, i_k is the spherical inertia of the k^{th} layer about its centroid G_k and m_k is its mass.

To formulate a closed-form expression for this sum, we use the terms defined in the previous section. For conciseness let vector $\overrightarrow{V_3} = \overrightarrow{G_L F}$.

For a swirl pattern (Sec. 7.3), the spherical inertia of the lattice L about its center of mass is:

$$i_L = q_0 + m_0(q_1 + q_2 + q_3 + q_4 + q_5) \tag{8.13}$$

where,

$$q_0 = i_0 \sum m_w^{5k},\ q_1 = |V_3|^2 \sum m_w^{3k}$$
$$q_2 = |\overrightarrow{V}|^2 \sum m_w^{5k},\ q_3 = 2V_3 \cdot V_1 \sum m_w^{4k}$$
$$q_4 = 2V_3 \cdot V_2 \sum (m_w^{4k} c_k),\ q_5 = 2V_3 \cdot V_2^{\circ}(\overrightarrow{N}) \sum (m_w^{4k} s_k)$$

For a screw pattern (Sec. 7.3), it is:

$$i_L = q_0 + m_0(q_1 + q_2 + q_3 + q_4 + q_5) \tag{8.14}$$

where,

$$q_0 = \underline{k} i_0,\ q_1 = \underline{k}(|\overrightarrow{V_3}|^2 + |\overrightarrow{V}|^2 + 2\overrightarrow{V_3} \cdot \overrightarrow{V_1})$$
$$q_2 = 2d(\overrightarrow{V} \cdot \overrightarrow{N} + \overrightarrow{V_3} \cdot \overrightarrow{N}) \sum k,\ q_3 = d^2 \sum k^2$$
$$q_4 = 2\overrightarrow{V_3} \cdot \overrightarrow{V_2} \sum c_k,\ q_5 = 2\overrightarrow{V_3} \cdot \overrightarrow{V_2}^{\circ}(\overrightarrow{N}) \sum s_k$$

Extensions of such closed-form expressions for inertia about a given axis and hence for inertia tensors are more challenging and are not addressed in this thesis.

8.2 Close approximation of mass properties of a hub

The first of the two challenges in computing mass properties of a large lattice is addressed by designing the lattice as a steady lattice and using the closed-form expressions provided in the previous section. The second challenge in computing the mass properties of a large lattice is the geometric complexity of intersections between the balls and beams of a lattice. One of the ways to address the above challenge is to to approximate the integral properties of a hub from its voxelization or octree decomposition [62], from ray sampling, or from a

triangulation of its boundary. For example, we can triangulate the surface of the ball of the hub and project (ray shooting) its vertices outwards from the ball's center C to the hub's surface, then approximate the hub's volume by summing the volume of the tetrahedron formed by each triangle with C. But even with large number of rays, this will produce large approximation errors (Fig. 8.1), especially towards the far end of the half-beam, away from center C.

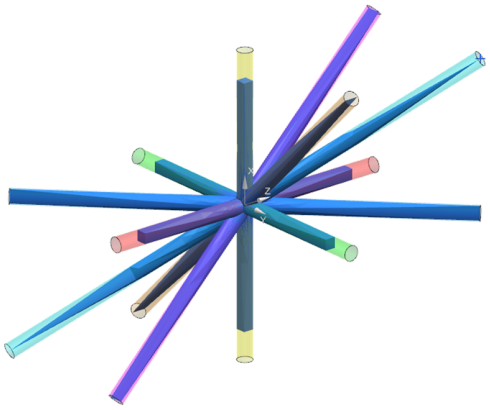


Figure 8.1: Discretizing entire hub (solid color) has large approximation error with respect to actual hub (transparent).

To improve hub's approximation with considerably fewer rays, we observe that for a clean lattice, i.e. for a lattice with disjoint hubs, the intersection between the balls and beams of the lattice is confined to individual hubs and in fact at each hub these intersections are confined within a minimal radius ***inflated ball*** that is concentric with the ball of the hub. The half-beams of the hub may overlap within this inflated ball and are disjoint outside of it. Furthermore, we assume that the length/diameter ratios of the beams are sufficiently large, so that the overlapping region between lattice elements is a very small fraction of the overall lattice. We conjecture that most lattices respect this assumption, since prior art [105, 106] recommends a ratio superior to 5, which ensures the accuracy of structural analysis of the lattices. We therefore estimate the integral properties of a hub as follows:

For each pair of half-beams of a hub, we compute an inflated ball (concentric with the ball of the hub) such that the two half-beams of that pair are disjoint outside that inflated

ball. The largest of these inflated balls for all pairs of half-beams in a hub is the inflated-ball for that hub. We then split the hub into a single ***core*** portion that is within this inflated ball and several ***clean*** portions that are outside the inflated ball, one for each of the half-beams (Fig. 8.2). We then approximate the integral properties of the hub by aggregating the integral properties of its core and clean portions.

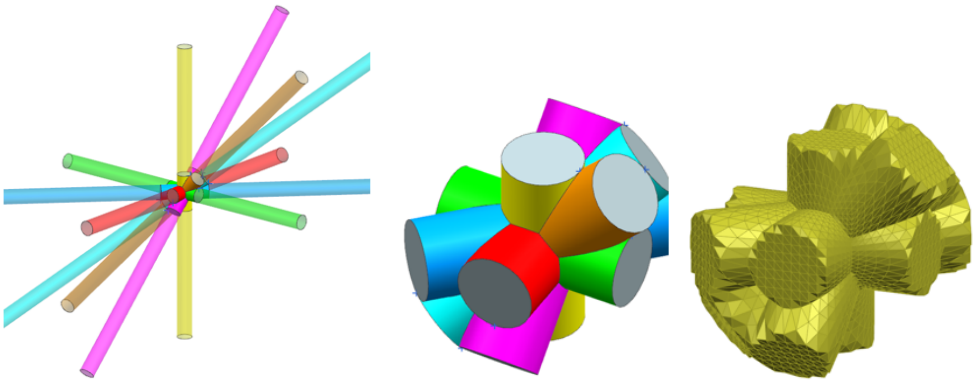


Figure 8.2: A hub is split into a core portion and clean portions of several beams (left). The core (center). Discretized core (right). Mass property of a hub is approximated by aggregating approximate value of that property for the core and exact values of that property for the clean portions.

The inflated ball of a hub is centered at the center O of the ball of that hub. The radius of this inflated ball is computed as follows. Consider a hub with n half-beams. Let $r_{i,j}$ be the radius of the smallest sphere $\mathbf{S}_{i,j}$ centered at O, outside of which the two half-beams with surfaces $\mathbf{Q}_i$ and $\mathbf{Q}_j$ don't interfere with each other. Then, the radius r of the inflated ball is obtained by

$$\begin{aligned} r &= \max r_{i,j}, \forall j \neq i \text{ AND } i,j \in [1,n] \\ \text{where, } r_{i,j} &= distance(O, point(\mathbf{C}[i][j], 0.5)) \end{aligned} \tag{8.15}$$

Where $point(\mathbf{C}[i][j], 0.5)$ is a point on $\mathbf{S}_{i,j}$. Computing this point is reduced to a 2D problem (in the plane of the axes of $\mathbf{Q}_i$ and $\mathbf{Q}_j$) of computing the intersection point of the profile curves of $\mathbf{Q}_i$ and $\mathbf{Q}_j$ and computing its distance from O. For cone half-beams this

is trivial, as the two profiles are straight lines. For quador half-beams, it may appear that we need to intersect two conics. However, we avoid computing this intersection by directly computing the point corresponding to parameter 0.5 on the consistently parameterized intersection curve $\mathbf{C}[i][j]$ between the two half-beams.

The integral properties of the core are approximated from its tetrahedral discretization using the ray shooting approach described above, while that of the clean portions (cylinders, cones, ellipsoids, hyperboloids) are computed using closed-form expressions. For quador beams these closed-form expressions are derived as follows.

Given a surface-of-revolution, defined by profile $y = f(x)$ and x-axis as its axis-of-revolution, its surface area [107] is given by:

$$S = 2\pi \int y\sqrt{1 + (dy/dx)^2}dx \tag{8.16}$$

For a quador surface the profile curve is given by $ax^2 + y^2 + 2bx + c = 0$. Upon differentiating we obtain $dy/dx = -(ax + b)/y$. Substituting it in (8.16) we obtain, $S = 2\pi \int \sqrt{y^2 + (ax + b)^2}dx$, using the equation of the profile curve we eliminate y from this equation to obtain, $S = 2\pi \int \sqrt{a(a-1)x^2 + 2b(a-1)x + b^2 - c}dx$. This is an integral of the form $\int \sqrt{Ax^2 + Bx + C}dx$, which has a standard closed form solution [108].

Similarly, the volume of a solid of revolution [109] is given by:

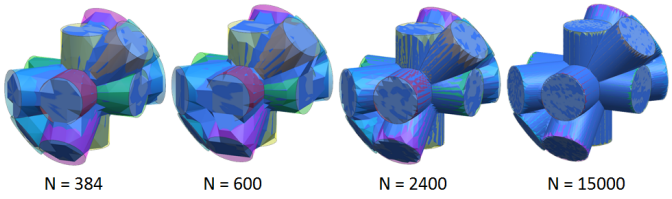
$$V = \pi \int y^2 dx \tag{8.17}$$

Which upon substituting y^2 and integrating gives, $V = -\pi(ax^3/3 + bx^2 + cx + constant)$.

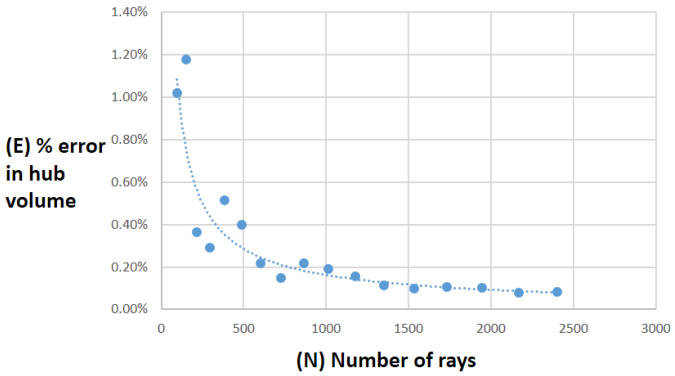
8.3 Experimental validations of our approach to estimate the mass properties of a steady lattice

To validate the efficiency of our approach to estimate the mass properties of a hub, we consider the hub of Fig. 8.2-left. This hubs has 14 half-beams. For simplicity, we consider

cone beams. We estimate the volume of this hub using Parasolid (Geometric Modeling Kernel). We consider the volume estimated by the Parasolid Kernel to be the true volume of the hub. We then estimate the volume of this hub by the approach described in (Sec. 8.2), for different number of rays (level of discretization of the core of the hub) and measure the error in estimating the true volume of the hub. Figure 8.3 shows that the volume error of the hub reduces quickly to a low value (0.22%) with just 600 rays per hub. In comparison this error is more than 2.5% if we discretize the entire hub using 600 rays.



(a) Discretized core inside the true core (transparent blue).

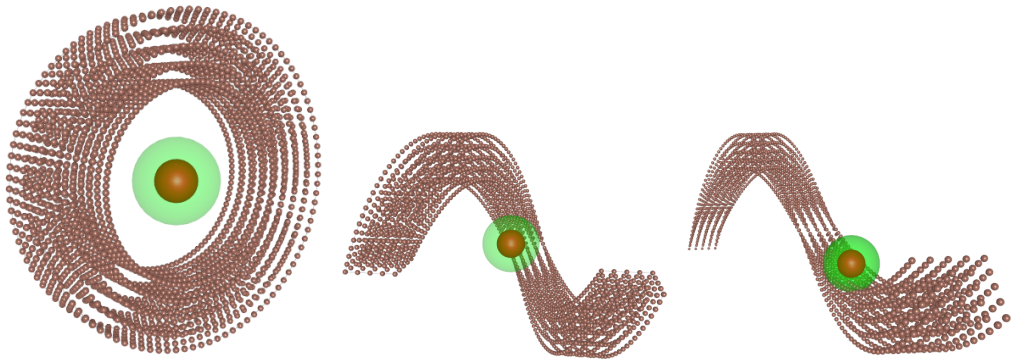


(b) Hub's volume error vs number of rays

Figure 8.3: Improved approximation at small number of rays due to hub splitting

To validate the closed-form expressions provided in Sec. 8.1, we perform the following experiments. Fig. 8.4 shows results of comparing volume and centroid of the cells of balls of a steady lattice (no beams), computed by using brute force versus using the closed-form expressions for volume Eq. 8.17, and centroid 8.7 and 8.11 of a steady lattice. The green ball (transparent) corresponds to closed-form results, its volume equals to that of all the balls of the lattice and is placed at the centroid of the centers of these balls. Similarly the red ball (solid color) corresponds to result from brute force computation. The red ball's radius has been reduced by half to visualize the two balls together. Observe that as

expected, the centers of the two ball are identical and the radius of the red ball is half of that of the green ball.



(a) pure rotation (b) screw pattern (c) swirl pattern

Figure 8.4: Verification of equations for volume and centroid.

Finally, we validate that our approach reduces the computational complexity to estimate the integral properties of a steady lattice of size $(\underline{i} \times \underline{j} \times \underline{k})$ from $O(\underline{i} \times \underline{j} \times \underline{k})$ to $O(\underline{i} \times \underline{j})$. To illustrate this acceleration, we consider the example of the lattice of Fig. 7.8 with repetition counts $3 \times 30 \times 20$ and $64,800$ beams in total. For each hub of the lattice, we approximate its volume using 600 rays per hub, following the approach described in Sec. 8.2. For brute force computation, we explicitly compute the volume of each hub in the lattice and sum them up to approximate the volume of the lattice. For closed form computation, we first explicitly compute the volume of all the hubs in the $k = 0$ layer of hubs, sum them up to approximate the volume v_0 of the $k = 0$ layer of hubs and then use the closed form expression of Eq. 8.17 to approximate the volume of the lattice. We report the average values of three consecutive runs. While, the brute force computation took $30,635$ milliseconds, the closed form computation took just $1,565$ milliseconds. That's a reduction by a factor of 19.57, as expected for $\underline{k} = 20$. The significance of such a reduction may be appreciated even more considering that the closed-form computation of the volume of all the beams of a ***bent and graded*** octahedral lattice (right image in Fig. 1.1) of size $10000 \times 10000 \times 10000$, with more than 24×10^{12} beams, took under 10 minutes, which if done by brute force would take several days.

In this chapter we described efficient ways to closely approximate the mass properties of a steady lattice. In the next chapter we discuss efficient ways to perform geometric queries, such as PMC, minimum distance, ray intersection, slicing and volume meshing on a lattice with quador beams.

CHAPTER 9

ANTICIPATED BENEFITS OF QUADOR BEAMS ON PROCESSING A HUB

To fabricate a lattice of quador-hubs, one may obtain a triangle mesh from the Brep of the hub, and send it to a 3D printer. We have a separate paper under review, where we discuss computing watertight mesh of a selected subset of hubs in a lattice. Further, we envision a future where 3D printers accept unions or voxels [110]. Towards that future, lattices with quador hubs provide several advantages. We take advantage of decomposing a clean lattice into an assembly of pair wise disjoint hubs, to distribute a query on the lattice to its hubs. In case the lattice is a steady lattice, we accelerate it further by identifying a small set of candidate hubs for a query and then distributing that query to these hubs. Efficiently identifying these candidate hubs is a topic of research of my colleague Kelsey Kurzeja ([15]). Here, I specifically discuss, efficient ways to perform a set of fundamental geometric queries on a single hub. These queries are useful for evaluating a lattice structure "as designed", and though not discussed in this thesis, may be useful in effectively considering manufacturing process related constraints.

9.1 Point Membership Classification (PMC)

Point-in-hub test can be is reduced to a point-in-ball test and several point-in-beam tests. While a point-in-ball test is trivial, each point-in-beam test is reduced to a point-in-circle test, in the plane passing through the point and perpendicular to the beam's axis. Having the CSG representation of the hub and the analytical expressions for the surfaces allows fast and accurate PMC.

9.2 Minimum distance query

In general, to measure the distance from a reference point P to the boundary of a solid, one has to cycle through all the faces, edges and vertices of the solid, measuring the distance of P from each of them, and then pick the closest projection.

For a quador hub, if P is an external point, then we only need to do the point-to-face calculations, and not the point-to-edge and point-to-vertex calculations. This is because there are no edges or vertices that are (strictly) convex. The point-to-face calculation is easier for quadors than for general quadrics, as it is reduced to a 2D problem of point-to-conic calculation [111] in the plane containing the point P and the axis of the quador.

If P is an internal point, then we do need to do all the point-to-edge and point-to-vertex calculations, in addition to the point-to-face ones. But on a quador hub, all edges are conics, and a point-to-conic calculation is easier than a point-to-spline calculation (though it's still fairly difficult).

9.3 Ray intersection

Computing the intersection of a ray with a hub using the Brep of the hub would require point-in-face tests on curved faces bounded by curved edges. These tests may be computationally expensive. We therefore suggest that we either intersect the ray with the CSG primitives and use the CSG expression of the hub to compute the actual intersections [112, 38], or we use the CST representation of the hub, wherein we first compute the candidate intersections of a ray with each of the quador surfaces and then test each of these candidates for containment inside the trimming polyhedron the corresponding surface. If the point is inside the trimming polyhedron then it is a valid intersection point.

9.4 Planar slicing

Given a plane **L**, we compute its intersection with a hub as follows. We first compute the intersection of plane **L** with each of the quador surfaces including that of the hub's ball [36]. As we have only quador surfaces, we get only conic intersection curves (Fig. 9.1). We then trim each intersection curve by the trimming polyhedron of the corresponding surface. Note that some intersection curves might be trimmed away completely as they lie outside the trimming polyhedron. Having a CST representation of the hub and quador surfaces for beams allows fast and accurate computation of planar slices.

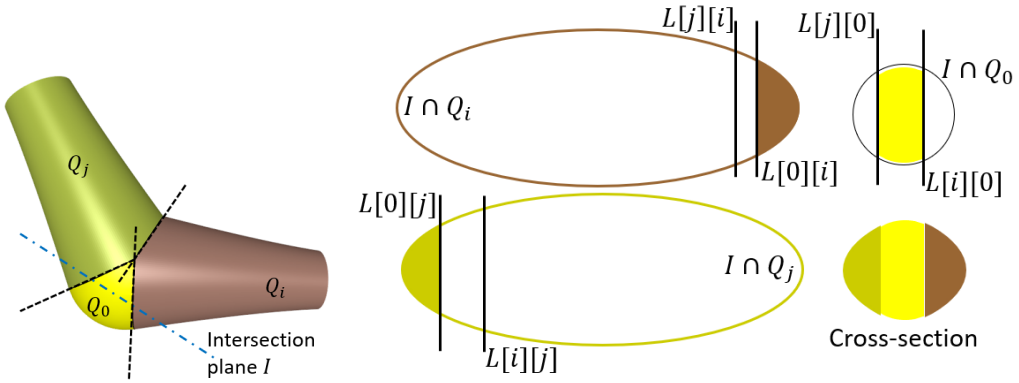


Figure 9.1: Slicing of a quador hub.

9.5 Volume decomposition

The CST representation of a hub decomposes it in to convex pieces. Each piece is the intersection of a quador half-space with multiple linear half-space. Fig. 9.2 This ***convex decomposition*** may be useful in several applications [113, 114], e.g. volume meshing, collision detection, and skeleton extraction.

9.6 Surface meshing

The Brep of a hub decomposes its surface into a spherical surface trimmed by loops of circular edges and several quador surfaces, each trimmed by loops of conic edges. These

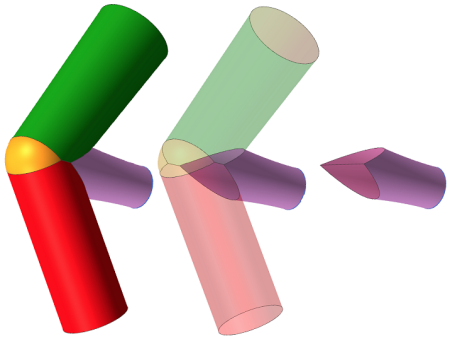


Figure 9.2: Decomposition of a hub into convex pieces. The rightmost subfigure shows the convex piece belonging to the magenta half-beam of the hub on the left.

edges may be useful in constraint meshing of each of the faces of the hub. For example, if we assume that a half-beam doesn't stick out of another (i.e. two half-beams don't intersect in a loop-edge), then the face of a half-beam will have a cylindrical topology with two loops, one on each of its ends. One of these two loops is a circle in the split-plane of the half-beam, and the other is either a circle in the contact-plane of the half-beam or is a loop of conic edges, each conic edge on the intersection of this half-beam's surface with the surface of the ball or another half-beam of the hub. This face may then be meshed following the natural parameterization of the underlying surface-of-revolution.

CHAPTER 10
SUMMARY AND CONCLUSIONS

Additive manufacturing (AM) or 3D printing is making it possible to fabricate parts with unprecedented structural complexity and optimized mechanical properties. It does so by allowing precise control over the distribution of material across the part. Hence, instead of a lump of material, engineers model the interior of a part (or the part itself) as a lattice of a *large number* (billions to trillions) of micro scale elements with *complex connectivity*. Often these elements are designed as beams that interconnect at nodes to form a lattice. Then by controlling the shape, size and connectivity of each of these beams and nodes, the lattices exhibit material properties that are superior to a homogeneous lump of material. Therefore, to realize the transformational promise of AM, the Solid Modeling community must address the challenges associated with the *design and processing* of such *large and complex* lattices.

Balancing design freedom and computational complexity is challenging, more so when the goal is to design a lattice of trillions of beams with complex connectivity. While we make several specific contributions to the design and processing of lattices, our underlying principle is to hit that *sweet spot*, wherein our solutions offer enough degrees of freedom for designing a useful class of lattices, and yet keep the computational cost at check.

We first focus on the design of lattice beams. To a very large extent, lattices are modeled with beams that are cylinders and cones, and each beam connects two nodes, that are modeled as spheres. To improve the structural performance, these beams are often modeled as ***cone beams*** [16], wherein the surface of a beam connects smoothly with that of the two balls connected by that beam. While controlling the position, size, and connectivity of each of the cone beams does provide a reasonable freedom to a designer, a potent way to expand this freedom is to let the designer control the shape of the beams. Towards that we propose

a family of beams called ***quador beams***. This family includes cone beams and offers a much larger variety of shapes than cone beams, and yet like a cone beam, each beam in this family is bound by a surface of only degree 2. We provide an intuitive scheme with simple expressions to construct and control the shape of a quador beam. In our scheme, the user controls the shape of a beam by just shrinking or growing a third ball in the middle of the two balls connected by that beam. Compared to a cone beam, whose shape is implicit from its two balls, a quador beam provides the necessary decoupling to control the shape of the beam independent of its two balls. Furthermore, the number of control parameters per beam is just one. Having less parameters to tweak makes it viable to fine tune them in an iterative procedure to optimize the lattice. We show that the quador beams give designers more control to optimize the strength/weight or surface-area/volume ratios of a lattice for intended applications.

We then focus on the lattices of quador beams. We may consider the solid model of a lattice as a union of balls and beams. While the intersection between two beams not incident on a ball of the lattice is undesirable, intersection between beams incident on the same ball is useful to strengthen the junction of beams forming at that ball, and to distribute the stresses evenly. With this understanding, we propose a new decomposition of the solid model of a lattice. We first split each beam into two half-beams, define a ***hub*** as the union of a ball and half-beams incident on that ball, and then decompose the lattice into an ***assembly of hubs***. Furthermore, we assume that the hubs in a lattice are pairwise disjoint, and we call such a lattice a ***clean lattice***. We discuss several advantages of this decomposition in processing a lattice of quador beams.

In a clean lattice, hubs are disjoint, but beams of a hub may intersect with the ball of the hub and with each other. Computing intersections of their surfaces reliably and accurately is a challenge. For example, two quadric surfaces in general intersect in degree 4 curves that are represented approximately by spline curves, leading to the well known problem of ”cracks” in solid models [115]. We report the *special property* of two quador half-beams

of a hub, that their surfaces intersect in conic, i.e. planar curves. This is remarkable, as for conic curves we have exact analytic and parametric representations, thus avoiding the problem of ”cracks”.

Another useful aspect of a clean lattice is that a query, e.g. PMC, may be distributed (for parallel processing) on its hubs. Furthermore, to accelerate querying on a single hub, we propose three representations of a hub, such that each of the representations is more suitable to perform certain queries. These three representations, namely the ***CSG***, the ***CST*** and the ***Brep*** of a hub are special in their own way. The CSG is a union of simple primitives, a sphere and quador half-beams, hence PMC queries can be distributed to these primitives. The CST is the union of interior disjoint ***chunks***, each chunk is a convex solid bounded by a quador surface and by planes separating it from other chunks. Such a decomposition is useful in volume meshing of the hub. Finally, the Brep has the exact geometry of the underlying curves and surfaces as opposed to their polygonal approximations. It has only quadric surfaces and conic edges, which are simpler to process than spline curves and surfaces. We provide the data structures and the algorithms to compute and query the geometry and the topology of each of these representations.

Though exact in terms of the geometry of the underlying curves and surfaces, the computation of the Brep of a hub may develop topological inaccuracies due to numerical round-off errors. We propose a simple and numerically robust method to compute the correct topology of a hub. In our method, we associate a ***polyhedral trimming complex*** with the hub. Each polyhedron of this complex trims the ball or a beam of the hub. We propose to use the topology of this complex in relation to the hub to infer the topology of the hub. In our simplified approach, we only deal with signs of numbers with non-nested square roots and we discuss efficient ways to compute them precisely. In short, we compute the boundary on each quadric surface as an arrangement of linear half-spaces on that surface, which is much simpler to compute than the arrangement of quadric half-spaces on a quadric surface.

Distributing a query over hubs may suffice for processing lattices with millions of hubs, but for lattices with several billions or trillions of hubs, we may need a way to process them without querying all of their hubs. For that we propose a special class of lattices called the ***steady lattices***. One of the special properties of a steady lattice is that its hubs are arranged into rows (patterns) such that hubs in a row are related by similarity transforms. Furthermore, each hub in a row is related to the previous hub in that row by the same similarity transform. We call such rows ***steady rows*** of hubs. This property may be exploited to identify a smaller set of candidate hubs per row, instead of performing a query on all the hubs in each of the rows. Accelerated identification of these candidate hubs is covered in a parallel research by my colleague Kelsey Kurzeja [116]. In this thesis, we provide closed-form expressions for accelerated computing of the mass properties, such as surface area, volume, and centroid of a steady lattice. This is crucial, as these lattices are often optimized to reduce the weight of the part, while maintaining the required strength. We experimentally demonstrate that for a lattice with $n \times n \times n$ hubs, we reduced computational complexity from $O(n^3)$ to $O(n^2)$. Finally, we suggest methods to efficiently perform fundamental geometric queries, such as PMC, minimum distance, or planar slicing on a hub, taking advantage of the special properties of quador beams and of the three representations (CSG, CST, Brep) of the hub.

Research reported in this thesis lay foundation for several new research opportunities in future. For example, ***bent beams***, i.e. beams with curved instead of straight spine. Recall that though a quador beam has a curved profile compared to the straight profile of a cone beam, it still has a straight spine (axis) like a cone beam. Hence bent beams will further expand the range of beams for lattices. One of the ways to create a bent beam is to model its bounding surface as a ***cyclide*** [117] that is tangent to three given spheres. Such beams may be useful in modeling woven lattice structures. Of course computing and processing intersection of two cyclide surfaces would be challenging.

Another research opportunity is to develop ***multi-level or hierarchical lattices*** [118,

119]. We may create a multi-level lattice by replacing each of the balls and beams of a coarse level lattice by a ball or beam shaped lattice thus creating a lattice with two levels, coarse and medium. We may repeat this process with the balls and beams of the medium level lattice, to create a lattice with three levels, coarse, medium and fine, and so on to create lattices with multiple levels. A possible way to create such a multi-level lattice is to start with a steady lattice with fine elements and define filters to remove its balls and beams in an ordered manner, so as to create a medium level pattern of fine lattice elements. Then repeat this process to create coarse pattern of medium pattern of fine lattice elements, and so on to create more levels. Multi-level lattices have shown mechanical properties [30, 120] superior to that of single level lattices. The challenges are to develop a simple and intuitive procedure to let the designer specify these filters for the indented patterns, and to process [121] such lattices efficiently.

One more research opportunity is the blending of sharp intersection between two quador beams of a hub. Fehmi Cirak and Malcolm Sabin has published a small note [122] on creating this blend with a quadric surface. While it certainly smoothens intersection between two beam surfaces, two blend surfaces may again intersect sharply. Hence, A possible extension to this work would be to recursively blend the sharp intersection between two blend surfaces. Other possible opportunities may include, design of lattices that deviate from being a steady lattice minimally to be conforming to a surface, accelerated slicing and tool path generation for 3D printing of lattices, closed-form computation of the mass properties a hub instead of a close approximation (Sec. 8.2).

The contributions made in this thesis may help engineers design lattices with a large variety of quador (curved profile) beams, define and construct these beams easily, represent and process lattices of these beams accurately, construct a wide range of enormously large steady lattices of quador beams, and process them fast. This research was carried out under DARPA's TRAnsformative DESign (TRADES) project. Together with our partners, namely Siemens, Xerox PARC, and University of Michigan, we developed foundational

math to support enormously large and complex lattices for AM, contributed to solving several of DARPA's challenge problems including some problems from NASA, and delivered several state of the art tools to DARPA. We hope that our contributions may bring about transformative changes in the design and processing of lattices to realize the transformative promise of AM.

Appendices

APPENDIX A

NOTATIONS USED IN THIS THESIS

- Scalars: $a, b, c, d, e...$
- Points/Vertices: P, V_i
- Vectors: $\overrightarrow{N}, \overrightarrow{P_1P_2}$
- Geometric entity: $\mathbf{S}, \mathbf{Q}, \mathbf{\Pi}$
- Map/Transformation: $\boldsymbol{M}$
- unit vectors: $\underline{\overrightarrow{V}}, \underline{\overrightarrow{FP}}$
- angle between $\overrightarrow{V_1}$ and $\overrightarrow{V_2}$: $\overrightarrow{V_1}\hat{}\overrightarrow{V_2}$
- angle from $\overrightarrow{V_1}$ to $\overrightarrow{V_2}$ around vector $\overrightarrow{N}$: $\overrightarrow{V_1}^{\hat{N}}\overrightarrow{V_2}$
- $\overrightarrow{V}$ rotated by α around $\overrightarrow{N}$: $\overrightarrow{V}^{\circ}(\alpha, \overrightarrow{N})$
- $\overrightarrow{V}$ rotate by $\pi/2$ around $\overrightarrow{N}$: $V^{\circ}(N)$
- 2D vector $\overrightarrow{V}$ rotated clockwise by $\pi/2$: $\overrightarrow{V}^{\circ}$
- Similarity frame (origin O, orthonormal basis vectors $\overrightarrow{I}$, $\overrightarrow{J}$, $\overrightarrow{K}$ and scaling s): $\mathbf{F}(O, \overrightarrow{I}, \overrightarrow{J}, \overrightarrow{K}, s)$

APPENDIX B

GENERALIZED BRIANCHON'S THEOREM

A well-known theorem of Brianchon (Sec. 3.9 in [123]) states that for a quadrilateral with an inscribed conic, the lines connecting the points of tangency on pairs of opposite edges and the two diagonals of the quadrilateral intersect in a common point. A more general result is given by Salmon in Art. 264 of [27]. Salmon extends Brianchon's results to the case of a conic quadrilateral, i.e. a quadrilateral whose opposite edges are segments of the same conic. Fig. B.1, illustrate Brianchon's original theorem (left) and its extension to conic quadrilaterals (right).

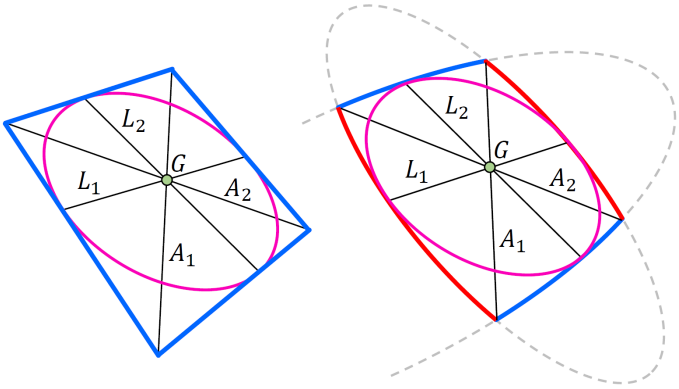


Figure B.1: Brianchon's theorem and its extension

APPENDIX C

DERIVATION OF THE PARAMETERS OF A SWIRL MOTION AND A SCREW MOTION

Given two similarity frames $\mathbf{F}_0 := \mathbf{F}(O_0, \overrightarrow{I_0}, \overrightarrow{J_0}, \overrightarrow{K_0}, s_0)$ and $\mathbf{F}_n := \mathbf{F}(O_n, \overrightarrow{I_n}, \overrightarrow{J_n}, \overrightarrow{K_n}, s_n)$, we compute the parameters describing the swirl-motion:

$$P(t) = F + m^t(\overrightarrow{FP_0})^\circ(t\alpha, \overrightarrow{N}) \tag{C.1}$$

defined by a special decomposition of the similarity $\boldsymbol{S} = \mathbf{F}_n \cdot \mathbf{F}_0^{-1}$ into the commutative product, $\boldsymbol{D} \cdot \boldsymbol{R}$, of a dilation $\boldsymbol{D}$, by a scaling factor m about fixed point F and a rotation $\boldsymbol{R}$, by angle α around an axis through F with tangent $\overrightarrow{N}$ as follows.

When a vector $\overrightarrow{V}$ is rotated about $\overrightarrow{N}$, its tip traces an arc in a plane perpendicular to $\overrightarrow{N}$. Hence, vector $\delta\overrightarrow{V} = \overrightarrow{V}^o(\alpha, \overrightarrow{N}) - \overrightarrow{V}$ lies in that plane, and so do the vectors $\overrightarrow{V}$ and $\overrightarrow{V}^o(\alpha, \overrightarrow{N})$. Hence, if frame $\mathbf{F}_0$ rotates to frame $\mathbf{F}_n$ about $\overrightarrow{N}$, the following three vectors $(\delta\overrightarrow{I}, \delta\overrightarrow{J}, \delta\overrightarrow{K})$ will lie in a plane perpendicular to $\overrightarrow{N}$:

$$\delta\overrightarrow{I} = \overrightarrow{I_n} - \overrightarrow{I_0},\ \delta\overrightarrow{J} = \overrightarrow{J_n} - \overrightarrow{J_0},\ \delta\overrightarrow{K} = \overrightarrow{K} - \overrightarrow{K_0} \tag{C.2}$$

Since one of them may be null, we compute N as:

$$\overrightarrow{N} = \underline{\delta\overrightarrow{I} \times \delta\overrightarrow{J} + \delta\overrightarrow{J} \times \delta\overrightarrow{K} + \delta\overrightarrow{K} \times \delta\overrightarrow{I}} \tag{C.3}$$

The angle of rotation of each of the three axes of the frame $\mathbf{F}_0$ is:

$$\alpha_1 = \overrightarrow{I_0}^{\hat{N}}\overrightarrow{I_n},\ \alpha_2 = \overrightarrow{J_0}^{\hat{N}}\overrightarrow{J_n},\ \alpha_3 = \overrightarrow{K_0}^{\hat{N}}\overrightarrow{K_n} \tag{C.4}$$

Since one of the values above may be zero, and knowing that the values of α_1, α_2 and α_3 are all in $[-\pi, \pi]$, we compute the angle of rotation α as:

$$\alpha = max(\alpha_1, \alpha_2, \alpha_3) \tag{C.5}$$

We compute the scaling factor m as:

$$m = s_n/s_0 \tag{C.6}$$

To compute the fixed point F, we define $\overrightarrow{V_0} = \overrightarrow{FO_0}$, $\overrightarrow{V_n} = \overrightarrow{FO_n}$ and $\overrightarrow{T} = \overrightarrow{O_0O_n}$. Hence, we have:

$$\overrightarrow{V_n} = \overrightarrow{V_0} + \overrightarrow{T} \tag{C.7}$$

We define $\overrightarrow{K_s} = \overrightarrow{N}$ and compute $\overrightarrow{I_s}$ and $\overrightarrow{J_s}$ forming an orthonormal triplet $(\overrightarrow{I_s}, \overrightarrow{J_s}, \overrightarrow{K_s})$. We then write $\overrightarrow{V_0}$ and its rotated version as follows :

$$\overrightarrow{V_0} = x\overrightarrow{I_s} + y\overrightarrow{J_s} + z\overrightarrow{K_s},\ and \tag{C.8}$$

$$\overrightarrow{V_0}^{\circ}(\alpha, \overrightarrow{N}) = (cx - sy)\overrightarrow{I_s} + (cy + sx)\overrightarrow{J_s} + (\overrightarrow{V_0} - x\overrightarrow{I_s} - y\overrightarrow{J_s}) \tag{C.9}$$

where,

$$x = \overrightarrow{V_0}.\overrightarrow{I_s},\ y = \overrightarrow{V_0}.\overrightarrow{J_s},\ z = \overrightarrow{V_0}.\overrightarrow{K_s},\ and$$

$$c = \cos(\alpha),\ s = \sin(\alpha)$$

As $\overrightarrow{V_n} = m\overrightarrow{V_0}^{\circ}(\alpha, \overrightarrow{N})$, we use Eq. C.7 and Eq. C.9 to solve for x and y. Further, $(\overrightarrow{V_n}\cdot \overrightarrow{K_s})/(\overrightarrow{V_0}\cdot \overrightarrow{K_s}) = m$, we use Eq. C.7 to solve for z:

$$\begin{aligned} x = \frac{u(mc-1)+v(ms)}{(mc-1)^2+(ms)^2},\ y = \frac{v(mc-1)-u(ms)}{(mc-1)^2+(ms)^2} \\ z = \frac{w}{m-1},\ where\ u = \overrightarrow{T}\cdot\overrightarrow{I_s}, v = \overrightarrow{T}\cdot\overrightarrow{J_s}, w = \overrightarrow{T}\cdot\overrightarrow{K_s} \end{aligned} \tag{C.10}$$

Then, the fixed point is:

$$F = O_0 - (x\overrightarrow{I_s} + y\overrightarrow{J_s} + z\overrightarrow{K_s}) \tag{C.11}$$

For a screw-motion:

$$P(t) = F + (td)\overrightarrow{N} + (\overrightarrow{FP_0})^{\circ}(t\alpha, \overrightarrow{N}) \tag{C.12}$$

$$d = \overrightarrow{T}\cdot\overrightarrow{N} \tag{C.13}$$

and a point F on the axis-of-rotation of the screw-motion is computed by substituting $m = 1$ in Eq. C.10 to compute x and y, and choosing an arbitrary value of z, and then using Eq. C.11.

REFERENCES

[1] A. Gupta, G. Allen, and J. Rossignac, "Quador: Quadric-of-revolution beams for lattices," *Computer-Aided Design*, vol. 102, pp. 160–170, 2018.

[2] ——, "Exact representations and geometric queries for lattice structures with quador beams," *Computer-Aided Design*, vol. 115, pp. 64–77, 2019.

[3] A. Gupta, K. Kurzeja, J. Rossignac, G. Allen, P. S. Kumar, and S. Musuvathy, "Programmed-lattice editor and accelerated processing of parametric program-representations of steady lattices," *Computer-Aided Design*, 2019.

[4] Y. Wu, A. Gupta, K. Kurzeja, and J. Rossignac, "Chocc: Convex hull of cospherical circles and applications to lattices," *Computer-Aided Design*, 2020.

[5] T. A. Schaedler, A. J. Jacobsen, A. Torrents, A. E. Sorensen, J. Lian, J. R. Greer, L. Valdevit, and W. B. Carter, "Ultralight metallic microlattices," *Science*, vol. 334, no. 6058, pp. 962–965, 2011.

[6] T. A. Schaedler and W. B. Carter, "Architected cellular materials," *Annual Review of Materials Research*, vol. 46, pp. 187–210, 2016.

[7] S. R. Musuvathy, *Method for creating three dimensional lattice structures in computer-aided design models for additive manufacturing*, US Patent 20150193559A1, 2015.

[8] E. Arisoy, L. Mirabella, S. R. Musuvathy, and A. Parlak, *Functional 3-d: Optimized lattice partitioning of solid 3-d models to control mechanical properties for additive manufacturing*, US Patent App. 15/269,634, 2017.

[9] S. Daynes, S. Feih, W. F. Lu, and J. Wei, "Optimisation of functionally graded lattice structures using isostatic lines," *Materials & Design*, 2017.

[10] B. Mirtich, "Fast and accurate computation of polyhedral mass properties," *Journal of graphics tools*, vol. 1, no. 2, pp. 31–50, 1996.

[11] H. Wang and D. W. Rosen, "Parametric modeling method for truss structures," in *ASME Computers and Information in Engineering Conference*, 2002.

[12] H. Wang, Y. Chen, and D. W. Rosen, "A hybrid geometric modeling method for large scale conformal cellular structures," in *ASME Computers and Information in Engineering Conference, Long Beach, CA, Sept*, 2005, pp. 24–28.

[13] J. Rossignac, A. Safonova, and A. Szymczak, "Edgebreaker on a corner table: A simple technique for representing and compressing triangulated surfaces," in *Hierarchical and geometrical methods in scientific visualization*, Springer Berlin Heidelberg, 2003, pp. 41–50.

[14] G. Damiand, "Linear cell complex," *CGAL User and Reference Manual*, vol. 4, 2000.

[15] K. Kurzeja and J. Rossignac, "Rangefinder: Accelerating ball-interference queries against steady lattices," *Computer-Aided Design*, 2019.

[16] N. Max, "Cone-spheres," *SIGGRAPH Comput. Graph.*, vol. 24, no. 4, pp. 59–62, Sep. 1990.

[17] 3MF Consortium - 3D Manufacturing Format Specifications, `https://3mf.io/beam-lattice-extension/`, accessed in April, 2020.

[18] R. N. Goldman, "Quadrics of revolution," *IEEE Computer Graphics and Applications*, vol. 3, no. 2, pp. 68–76, 1983.

[19] N. Max, "Cone-spheres," in *ACM SIGGRAPH Computer Graphics*, ACM, vol. 24, 1990, pp. 59–62.

[20] J. Jia, A. Joneja, and K. Tang, "Revolute quadric decomposition of canal surfaces and its applications," *Computational Science–ICCS 2005*, pp. 95–104, 2005.

[21] T. Gossling, "Bulge, shear and squash: A representation for the general conic arc," *Computer-Aided Design*, vol. 13, no. 2, pp. 81 –84, 1981.

[22] J. R. Rossignac and A. A. Requicha, "Piecewise-circular curves for geometric modeling," *IBM Journal of Research and Development*, vol. 31, no. 3, pp. 296–313, 1987.

[23] A. Kurnosenko, "Biarcs and bilens," *Computer Aided Geometric Design*, vol. 30, no. 3, pp. 310–330, 2013.

[24] A. Nutbourne and R. Martin, *Differential Geometry Applied to Curve and Surface Design: Foundations*. E. Horwood, 1988.

[25] A. Glassner, "Globs: A primitive shape for graceful blends between circles," *Journal of Computer Graphics Techniques Vol*, vol. 4, no. 3, 2015.

[26] E.-W. Chionh, "Shifting planes always implicitize a surface of revolution," *Computer Aided Geometric Design*, vol. 26, no. 4, pp. 369 –377, 2009, Geometric Mod-

eling and Processing 2008, 5th International Conference on Geometric Modeling and Processing.

[27] G. Salmon, *A Treatise on Conic Sections*. Longman, Green, Brown, and Longmans, 3rd Edition,1855.

[28] L. Zhang, S. Feih, S. Daynes, Y. Wang, M. Y. Wang, J. Wei, and W. F. Lu, "Buckling optimization of kagome lattice cores with free-form trusses," *Materials & Design*, 2018.

[29] C. S. Roper, K. D. Fink, S. T. Lee, J. A. Kolodziejska, and A. J. Jacobsen, "Anisotropic convective heat transfer in microlattice materials," *AIChE Journal*, vol. 59, no. 2, pp. 622–629, 2013.

[30] L. R. Meza, A. J. Zelhofer, N. Clarke, A. J. Mateos, D. M. Kochmann, and J. R. Greer, "Resilient 3d hierarchical architected metamaterials," *Proceedings of the National Academy of Sciences*, vol. 112, no. 37, pp. 11 502–11 507, 2015.

[31] L. Dupont, D. Lazard, S. Lazard, and S. Petitjean, "Near-optimal parameterization of the intersection of quadrics: 1,2,3. parameterizing singular intersections," *Journal of Symbolic Computation*, vol. 43, no. 3, pp. 168–232, 2008.

[32] G. Salmon, *A treatise on the analytic geometry of three dimensions*. Hodges, Smith, and Company, 7th Edition, Vol. 1, 1927, Art. 140 at page 137.

[33] C.-K. Shene, "Planar intersection, common inscribed sphere and dupin blending cyclides," in *Proceedings on the second ACM symposium on Solid modeling and applications*, ACM, 1993, pp. 487–488.

[34] C. M. Hoffmann and J. R. Rossignac, "A road map to solid modeling," *IEEE Transactions on visualization and computer graphics*, vol. 2, no. 1, pp. 3–10, 1996.

[35] K.-J. Kim, M.-S. Kim, and R. Martin, "Intersecting a translationally or rotationally swept surface with a plane or a sphere," in *Proc. of Korea Israel Bi-National Conference on Geometrical Modeling and Computer Graphics in the World Wide Web Era*, 1999, pp. 165–193.

[36] C.-K. Shene and J. K. Johnstone, "Computing the intersection of a plane and a revolute quadric," *Computers & graphics*, vol. 18, no. 1, pp. 47–59, 1994.

[37] J. Rossignac and A. Requicha, "Solid modeling," in *The Encyclopedia of Electrical and Electronics Engineering, Ed. J. Webster*, John Wiley Sons., 1999.

[38] J. Hable and J. Rossignac, "CST: Constructive solid trimming for rendering BReps and CSG," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 13, no. 5, pp. 1004–1014, 2007.

[39] C. M. Brown, "Padl-2: A technical summary," *IEEE Computer Graphics and Applications*, no. 2, pp. 69–84, 1982.

[40] R. F. Sarraga, "Algebraic methods for intersections of quadric surfaces in gmsolid," *Computer Vision, Graphics, and Image Processing*, vol. 22, no. 2, pp. 222–238, 1983.

[41] V. Shapiro, "Solid modeling," *Handbook of computer aided geometric design*, vol. 20, pp. 473–518, 2002.

[42] I. Stroud, *Boundary representation modelling techniques*. Springer Science & Business Media, 2006.

[43] I. Stroud and H. Nagy, *Solid modelling and CAD systems: how to survive a CAD system*. Springer Science & Business Media, 2011.

[44] M. Mantyla, *Introduction to Solid Modeling*. New York, NY, USA: W. H. Freeman & Co., 1988, ISBN: 0-88175-108-1.

[45] C. M. Hofmann, *Geometric and solid modeling: an introduction*. Morgan Kaufmann, 1989.

[46] J. Rossignac and H. Voelcker, "Active zones in CSG for accelerating boundary evaluation, redundancy elimination, interference detection, and shading algorithms," *ACM Transactions on Graphics (TOG)*, vol. 8, no. 1, pp. 51–87, 1989.

[47] A. A. Requicha and H. B. Voelcker, "Boolean operations in solid modeling: Boundary evaluation and merging algorithms," *Proceedings of the IEEE*, vol. 73, no. 1, pp. 30–44, 1985.

[48] S. Krishnan, D. Manocha, M Gopi, T. Culver, and J. Keyser, "Boole: A boundary evaluation system for boolean combinations of sculptured solids," *International Journal of Computational Geometry & Applications*, vol. 11, no. 01, pp. 105–144, 2001.

[49] M. O. Benouamer and D. Michelucci, "Bridging the gap between csg and brep via a triple ray representation," in *Proceedings of the fourth ACM symposium on Solid modeling and applications*, ACM, 1997, pp. 68–79.

[50] J. Keyser, T. Culver, M. Foskey, S. Krishnan, and D. Manocha, "Esolid—a system for exact boundary evaluation," *Computer-Aided Design*, vol. 36, no. 2, pp. 175–193, 2004.

[51] M. O. Benouamer, D. Michelucci, and B. Peroche, "Error-free boundary evaluation based on a lazy rational arithmetic: A detailed implementation," *Computer-Aided Design*, vol. 26, no. 6, pp. 403–416, 1994.

[52] K. Sugihara, "A robust and consistent algorithm for intersecting convex polyhedra," in *Computer Graphics Forum*, Wiley Online Library, vol. 13, 1994, pp. 45–54.

[53] R. P. Banerjee and J. Rossignac, "Topologically exact evaluation of polyhedra defined in CSG with loose primitives," *Computer Graphics Forum*, vol. 15, no. 4, pp. 205–217, 1996.

[54] D. J. Jackson, "Boundary representation modelling with local tolerances," in *Proceedings of the Third ACM Symposium on Solid Modeling and Applications*, ser. SMA '95, Salt Lake City, Utah, USA: ACM, 1995, pp. 247–254, ISBN: 0-89791-672-7.

[55] E. Berberich, M. Hemmer, L. Kettner, E. Schömer, and N. Wolpert, "An exact, complete and efficient implementation for computing planar maps of quadric intersection curves," in *Proceedings of the twenty-first annual symposium on Computational geometry*, ACM, 2005, pp. 99–106.

[56] S. Lazard, L. M. Peñaranda, and S. Petitjean, "Intersecting quadrics: An efficient and exact implementation," *Computational Geometry*, vol. 35, no. 1-2, pp. 74–99, 2006.

[57] L. Dupont, D. Lazard, S. Lazard, and S. Petitjean, "Near-optimal parameterization of the intersection of quadrics: I, II, III," *Journal of Symbolic Computation*, vol. 43, no. 3, pp. 168–232, 2008.

[58] M. Hemmer, L. Dupont, S. Petitjean, and E. Schömer, "A complete, exact and efficient implementation for computing the edge-adjacency graph of an arrangement of quadrics," *Journal of Symbolic Computation*, vol. 46, no. 4, pp. 467–494, 2011.

[59] F. Klein, "A new approach to point membership classification in b-rep solids," in *IMA International Conference on Mathematics of Surfaces*, Springer, 2009, pp. 235–250.

[60] S. D. Roth, "Ray casting for modeling solids," *Computer graphics and image processing*, vol. 18, no. 2, pp. 109–144, 1982.

[61] J. Rossignac, A. Megahed, and B.-O. Schneider, "Interactive inspection of solids: Cross-sections and interferences," in *ACM SIGGRAPH Computer Graphics*, ACM, vol. 26, 1992, pp. 353–360.

[62] Y. T. Lee and A. A. Requicha, "Algorithms for computing the volume and other integral properties of solids - i and ii," *Communications of the ACM*, vol. 25, no. 9, pp. 635–641, 1982.

[63] J. R. Rossignac and A. A. Requicha, "Offsetting operations in solid modelling," *Computer Aided Geometric Design*, vol. 3, no. 2, pp. 129–148, 1986.

[64] J. Levin, "A parametric algorithm for drawing pictures of solid objects composed of quadric surfaces," *Communications of the ACM*, vol. 19, no. 10, pp. 555–563, 1976.

[65] W. Wang, B. Joe, and R. Goldman, "Computing quadric surface intersections based on an analysis of plane cubic curves," *Graphical Models*, vol. 64, no. 6, pp. 335–367, 2002.

[66] E.-W. Chionh, R. N. Goldman, and J. R. Miller, "Using multivariate resultants to find the intersection of three quadric surfaces," *ACM Transactions on Graphics (TOG)*, vol. 10, no. 4, pp. 378–400, 1991.

[67] P. Krysl and M. Ortiz, "Extraction of boundary representation from surface triangulations," *International journal for numerical methods in engineering*, vol. 50, no. 7, pp. 1737–1758, 2001.

[68] J. Rossignac, "Solid and physical modeling," in, Wiley Online Library, 2007.

[69] E. Brisson, "Representation of d-dimensional geometric objects," PhD thesis, 1990.

[70] C. M. Hoffmann, "Geometric and solid modeling," 1989.

[71] C. M. Hoffmann, "The problems of accuracy and robustness in geometric computation," *Computer*, vol. 22, no. 3, pp. 31–39, 1989.

[72] K. Sugihara and M. Iri, "A solid modelling system free from topological inconsistency," *Journal of Information Processing*, vol. 12, no. 4, pp. 380–393, 1989.

[73] D. Salesin, J. Stolfi, and L. Guibas, "Epsilon geometry: Building robust algorithms from imprecise computations," in *Proceedings of the fifth annual symposium on Computational geometry*, 1989, pp. 208–217.

[74] C. M. Hoffmann, J. E. Hopcroft, and M. Karasick, "Robust set operations on polyhedral solids," *IEEE Computer Graphics and Applications*, vol. 9, no. 6, pp. 50–59, 1989.

[75] J. Keyser, S. Krishnan, and D. Manocha, "Efficient and accurate b-rep generation of low degree sculptured solids using exact arithmetic: I—representations, ii-computations," *Computer Aided Geometric Design*, vol. 16, no. 9, pp. 841–859, 1999.

[76] C.-K. Yap, "Towards exact geometric computation," *Computational Geometry*, vol. 7, no. 1-2, pp. 3–23, 1997.

[77] D. Manocha and J. F. Canny, "Multipolynomial resultant algorithms," *Journal of Symbolic Computation*, vol. 15, no. 2, pp. 99–122, 1993.

[78] R. B. Kearfott, "Interval computations: Introduction, uses, and resources," *Euromath Bulletin*, vol. 2, no. 1, pp. 95–112, 1996.

[79] G. Damiand and P. Lienhardt, *Combinatorial maps: efficient data structures for computer graphics and image processing*. CRC Press, 2014.

[80] K. A. Ohori, G. Damiand, and H. Ledoux, "Constructing an n-dimensional cell complex from a soup of (n- 1)-dimensional faces," in *International Conference on Applied Algorithms*, Springer, 2014, pp. 37–48.

[81] G. Damiand and M. Teillaud, "A generic implementation of dd combinatorial maps in cgal," *Procedia Engineering*, vol. 82, pp. 46–58, 2014.

[82] M. I. Karavelas, "Exact geometric and algebraic computations in cgal," in *International Congress on Mathematical Software*, Springer, 2010, pp. 96–99.

[83] I. Sutherland and L. Laboratory, *Sketchpad: A Man-machine Graphical Communication System*, ser. Reports // MIT. Massachusetts Institute of Technology, Lincoln Laboratory, 1963.

[84] J. Rossignac, P. Borrel, and L. Nackman, "Procedure models for design and fabrication.," *Automation in the Design and Manufacture of large Marine Systems. Proceedings of the 16th annual MIT Sea Grant College Program Lecture and Seminar, 1989, Hemisphere Publishing Corp., New York (USA), 1990*, pp. 147–178,

[85] ——, "Interactive design with sequences of parameterized transformations," *Intelligent CAD Systems*, vol. 2, pp. 93–125, 1989.

[86] M. Van Emmerik, A. Rappoport, and J. Rossignac, "Simplifying interactive design of solid models: A hypertext approach," *The Visual Computer*, vol. 9, no. 5, pp. 239–254, 1993.

[87] J. Jang and J. Rossignac, "Octor: Subset selection in recursive pattern hierarchies," *Graphical Models*, vol. 71, no. 2, pp. 92–106, 2009.

[88] A. Pasko, O. Fryazinov, T. Vilbrandt, P.-A. Fayolle, and V. Adzhiev, "Procedural function-based modelling of volumetric microstructures," *Graphical Models*, vol. 73, no. 5, pp. 165–181, 2011.

[89] C. S. Daily, D. A. Lees, and D. D. McKitterick, *Truss structure design*, US Patent 6,170,560, 2001.

[90] J. Nguyen, S Park, D. W. Rosen, L. Folgar, and J. Williams, "Conformal lattice structure design and fabrication," in *Solid Freeform Fabrication Symposium, Austin, TX*, 2012, pp. 138–161.

[91] G. Elber, "Precise construction of micro-structures and porous geometry via functional composition," in *International Conference on Mathematical Methods for Curves and Surfaces*, Springer, 2016, pp. 108–125.

[92] M. A. Aguilovalentin, L. L. Beghini, B. W. Clark, W. R. Quadros, J. Robbins, B. Sneed, and T. E. Voth, "'PLATO" environment for designing with topology optimization.," Sandia National Laboratories (SNL-NM), Albuquerque, NM (United States); Sandia National Laboratories, Livermore, CA, Tech. Rep., 2015.

[93] E. B. Arisoy, S. Musuvathy, L. Mirabella, and E. Slavin, "Design and topology optimization of lattice structures using deformable implicit surfaces for additive manufacturing," in *ASME 2015 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, American Society of Mechanical Engineers, 2015, V004T05A003–V004T05A003.

[94] A. Biswas, V. Shapiro, and I. Tsukanov, "Heterogeneous material modeling with distance fields," *Computer Aided Geometric Design*, vol. 21, no. 3, pp. 215–242, 2004.

[95] Y. Chen, "3d texture mapping for rapid manufacturing," *Computer-Aided Design and Applications*, vol. 4, no. 6, pp. 761–771, 2007.

[96] A. Aremu, J. Brennan-Craddock, A. Panesar, I. Ashcroft, R. J. Hague, R. D. Wildman, and C. Tuck, "A voxel-based method of constructing and skinning conformal and functionally graded lattice structures suitable for additive manufacturing," *Additive Manufacturing*, vol. 13, pp. 1–13, 2017.

[97] M. F. Ashby and T. Lu, "Metal foams: A survey," *Science in China Series B: Chemistry*, vol. 46, no. 6, pp. 521–532, 2003.

[98] X. Liu and V. Shapiro, "Sample-based synthesis of functionally graded material structures," *Journal of Computing and Information Science in Engineering*, vol. 17, no. 3, p. 031 012, 2017.

[99] J. Martínez, J. Dumas, and S. Lefebvre, "Procedural voronoi foams for additive manufacturing," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, p. 44, 2016.

[100] J. Martínez, H. Song, J. Dumas, and S. Lefebvre, "Orthotropic k-nearest foams for additive manufacturing," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 121, 2017.

[101] J. Kim and J. Rossignac, "Screw motions for the animation and analysis of mechanical assemblies," *JSME International Journal Series C*, vol. 44, no. 1, pp. 156–163, 2001.

[102] K. N. Boyadzhiev, "Spirals and conchospirals in the flight of insects," *The College Mathematics Journal*, vol. 30, no. 1, p. 23, 1999.

[103] I. S. A. Padula, J. Benzing, and V. M. Asnani, *Superelastic tire*, US Patent 10,449,804, 2019.

[104] W. R. Inc., *Mathematica, Version 11.2*, Champaign, IL, 2017.

[105] C. Wang, L Berhan, and A. Sastry, "Structure, mechanics and failure of stochastic fibrous networks: Part i—microscale considerations," *Journal of Engineering Materials and Technology*, vol. 122, no. 4, pp. 450–459, 2000.

[106] L. Valdevit, S. W. Godfrey, T. A. Schaedler, A. J. Jacobsen, and W. B. Carter, "Compressive strength of hollow microlattices: Experimental characterization, modeling, and optimal design," *Journal of Materials Research*, vol. 28, no. 17, pp. 2461–2473, 2013.

[107] E. W. Weisstein, *"surface of revolution." from mathworld - a wolfram web resource*.

[108] I. S. Gradshteyn and I. M. Ryzhik, *Table of integrals, series, and products*. Academic press, 2014.

[109] E. W. Weisstein, *"solid of revolution." from mathworld - a wolfram web resource*, http://mathworld.wolfram.com/SolidofRevolution.html.

[110] J. Hiller and H. Lipson, "Design and analysis of digital materials for physical 3d voxel printing," *Rapid Prototyping Journal*, vol. 15, no. 2, pp. 137–149, 2009.

[111] N Chernov and S Wijewickrema, "Algorithms for projecting points onto conics," *Journal of Computational and Applied Mathematics*, vol. 251, pp. 8–21, 2013.

[112] J. Hable and J. Rossignac, "Blister: Gpu-based rendering of boolean combinations of free-form triangulated shapes," in *ACM Transactions on Graphics (TOG)*, ACM, vol. 24, 2005, pp. 1024–1031.

[113] J.-M. Lien, *Approximate convex decomposition and its applications, PhD-Thesis*, 01. 2006, vol. 69.

[114] J.-M. Lien and N. M. Amato, "Approximate convex decomposition of polyhedra," in *Proceedings of the 2007 ACM symposium on Solid and physical modeling*, ACM, 2007, pp. 121–131.

[115] J. Rossignac, "Through the cracks of the solid modeling milestone," in *From Object Modelling to Advanced Visual Communication*, Springer Berlin Heidelberg, 1994, pp. 1–75.

[116] K. Kurzeja and J. Rossignac, "Rangefinder: Accelerating ball-interference queries against steady lattices," *GVU Center Technical Reports, Georgia Institute of Technology*, 2018.

[117] Y. Srinivas and D. Dutta, "Intuitive procedure for constructing geometrically complex objects using cyclides," 1994.

[118] X. Liu and V. Shapiro, "Multiscale shape–material modeling by composition," *Computer-Aided Design*, vol. 102, pp. 194–203, 2018.

[119] F. Massarwi, J. Machchhar, P. Antolin, and G. Elber, "Hierarchical, random and bifurcation tiling with heterogeneity in micro-structures construction via functional composition," *Computer-Aided Design*, vol. 102, pp. 148–159, 2018.

[120] A. Vigliotti and D. Pasini, "Mechanical properties of hierarchical lattices," *Mechanics of Materials*, vol. 62, pp. 32–43, 2013.

[121] R. Lakes, "Materials with structural hierarchy," *Nature*, vol. 361, no. 6412, p. 511, 1993.

[122] F. Cirak and M. Sabin, "Adding quadric fillets to quador lattice structures," *Computer-Aided Design*, vol. 118, p. 102 754, 2020.

[123] H. S. M. Coxeter and S. L. Greitzer, *Geometry revisited*. Maa, 1967, vol. 19.